

AFRL-IF-WP-TR-2004-1545

**POWER AWARE SMART
SUBMUNITIONS (PASS)**

Jeffrey A. Barnett

Northrop Grumman Corporation
Integrated Systems
One Hornet Way, MS-9L72/W6
El Segundo, CA 90245-2804



JUNE 2004

Final Report for 21 May 2002 – 30 June 2004

Approved for public release; distribution is unlimited.

STINFO FINAL REPORT

© 2002 Kluwer

Appendix [2], resulting from Department of the Air Force contract number F33615-02-C-4001, is a draft of Chapter 1 in *Power Aware Computing*, Kluwer, 2002. The United States has for itself and others acting on its behalf an unlimited, nonexclusive, irrevocable, paid-up royalty-free worldwide license to use for its purposes.

**INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334**

NOTICE

USING GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA INCLUDED IN THIS DOCUMENT FOR ANY PURPOSE OTHER THAN GOVERNMENT PROCUREMENT DOES NOT IN ANY WAY OBLIGATE THE US GOVERNMENT. THE FACT THAT THE GOVERNMENT FORMULATED OR SUPPLIED THE DRAWINGS, SPECIFICATIONS, OR OTHER DATA DOES NOT LICENSE THE HOLDER OR ANY OTHER PERSON OR CORPORATION; OR CONVEY ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY RELATE TO THEM.

THIS REPORT IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS, IT WILL BE AVAILABLE TO THE GENERAL PUBLIC, INCLUDING FOREIGN NATIONALS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

/s/

KERRY L. HILL
Project Engineer
Embedded Information Systems Branch
Advanced Computing Division

/s/

ALFRED J. SCARPELLI
Team Leader
Embedded Information Systems Branch
Advanced Computing Division

/s/

JAMES S. WILLIAMSON, Chief
Embedded Information Systems Branch
Advanced Computing Division
Information Directorate

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> <i>OMB No. 0704-0188</i>	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YY) June 2004		2. REPORT TYPE Final		3. DATES COVERED (From - To) 5/21/2002 – 06/30/2004		
4. TITLE AND SUBTITLE POWER AWARE SMART SUBMUNITIONS (PASS)				5a. CONTRACT NUMBER F33615-02-C-4001		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER 62301E		
6. AUTHOR(S) Jeffrey A. Barnett				5d. PROJECT NUMBER M761		
				5e. TASK NUMBER 40		
				5f. WORK UNIT NUMBER 01		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) <div style="display: flex; justify-content: space-between;"> <div>Northrop Grumman Corporation Integrated Systems One Hornet Way, MS-9L72/W6 El Segundo, CA 90245-2804</div> <div>DARPA/IPTO Mr. Robert Graybill 3701 Fairfax Drive Arlington, VA 22203-1714</div> </div>				8. PERFORMING ORGANIZATION REPORT NUMBER 04-040		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Information Directorate Air Force Research Laboratory Air Force Materiel Command Wright-Patterson AFB, OH 45433-7334				10. SPONSORING/MONITORING AGENCY ACRONYM(S) AFRL/IFTA		
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-IF-WP-TR-2004-1545		
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.						
13. SUPPLEMENTARY NOTES © 2002 Kluwer. Appendix [2], resulting from Department of the Air Force contract number F33615-02-C-4001, is a draft of Chapter 1 in Power Aware Computing, Kluwer, 2002. The United States has for itself and others acting on its behalf an unlimited, nonexclusive, irrevocable, paid-up royalty-free worldwide license to use for its purposes.						
14. ABSTRACT The Power-Aware Smart Submunitions (PASS) project is funded through the DARPA IPTO Power Aware Computing and Communications (PACC) program. PASS has investigated the application of power-aware technology to military systems in general and smart submunitions in particular. This final report presents a summary of that investigation as well as several research notes included in the appendix. The research notes include a technology roadmap, description of how applications must interact with the power-management infrastructure, and methods to manage energy versus delay tradeoffs. Our conclusion is that application developers must determine the connections between energy management and optimization of mission goals; technology that performs energy conservation is valuable but, by itself, is not competent to optimally achieve mission objectives.						
15. SUBJECT TERMS Power Aware Systems, Smart Submunitions, Energy Management						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT: SAR	18. NUMBER OF PAGES 160	19a. NAME OF RESPONSIBLE PERSON (Monitor) Kerry Hill	
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (Include Area Code) (937) 255-6548 x3604	

Contents

1	Introduction	1
2	Background	3
3	Contractor Work Summary	7
3.1	Networking with Other Programs	7
3.2	CDRL Items	8
3.2.1	Technology Roadmap	8
3.3	Research Investigations	11
3.3.1	Application-Level Reasoning	11
3.3.2	Computation vs Communication	12
3.3.3	Redundancy Management	12
3.3.4	Energy and Time Management	13
4	Lessons Learned	16
A	Appendix: Research Notes	17

1 Introduction

The American military, during the previous century, made a radical transition from soldiers, primitive vehicles, and bullets to electronic warfare. While soldiers and bullets still dominate our images of war, the reality is a battlefield comprised of sophisticated platforms and a colossal support infrastructure built from communications, sensors, and computing. These factors are force multipliers as well as enablers for intelligent war-fighting. The electronics revolution has reached the point where we commonly see the development of autonomous systems that will, in part, remove the man from the battlefield.

Some of these autonomous systems must survive for long periods of time in hostile environments to achieve their missions. Unattended sensor networks, space systems, and long-endurance reconnaissance aircraft are some examples. A key component to their survival is energy management. These systems must either live on their onboard resources or scavenge energy from the environment. In either case, the problem is achieving maximum benefit, as measured by mission metrics, given extant energy constraints.

The exponential explosion in computing and communication capability, described by Moore's Law, poses other energy management problems: heat dissipation and packing density. As the feature size of components continues to shrink, the energy density grows. We are now at the point where many *home* computers employ liquid cooling. Some recent aircraft designs were forced to limit the number of boxes per rack—a form-factor crisis—because electrical increased energy consumption per component entailed vastly increased heat densities.

The Northrop Grumman Corporation (NGC) Power Aware Smart Submunitions (PASS) program has been funded to examine power management problems relevant to military applications with particular emphasis on smart submunitions. Figure 1 is a program quad chart. The work products are to include a technology roadmap and related research investigations. PASS is supported through the DARPA IPTO Power Aware Computing and Communications (PACC) program. Phase I PACC work at NGC was sponsored through a subcontract with USC/ISI East. The Phase II PASS program provides funds directly to NGC through AFRL.

Section 2 discusses the problem space of military aviation and the particular missions and technologies that need sophisticated power management technology. The contributions of the NGC PASS program are then summarized in Section 3. Section 4 discusses lessons learned from PASS. The

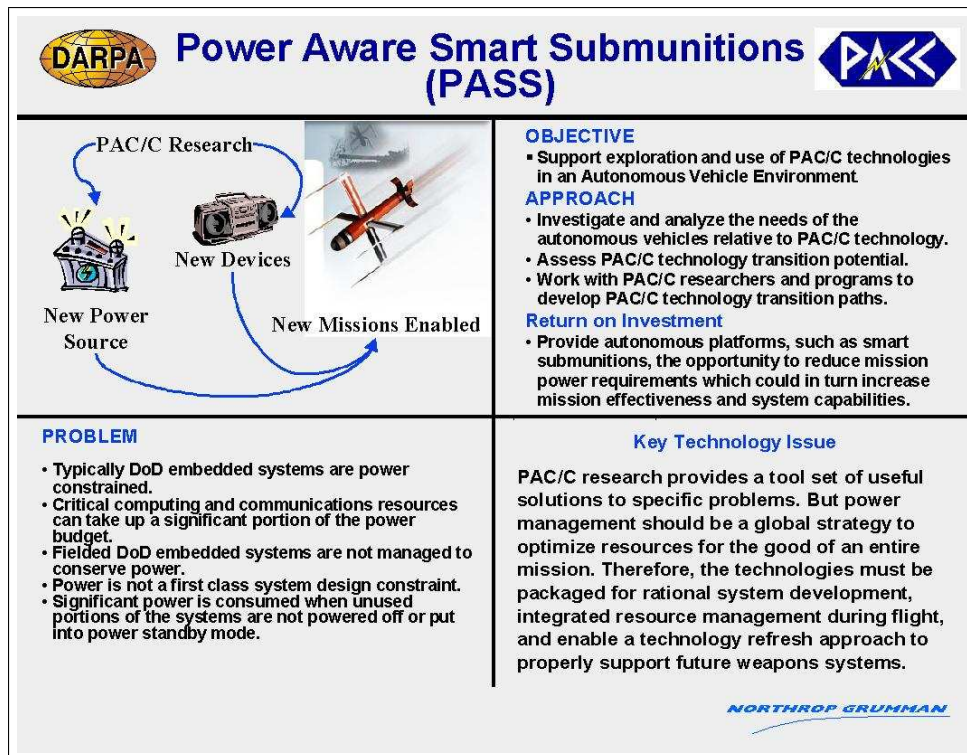


Figure 1: PASS project quad chart.

most important are that (1) power awareness is a system-level problem and (2) application developers are implicated in power management—the problems cannot be masked by a sophisticated API; rather, application knowledge and value systems are the keys to mission optimizations.

Appendix A contains the technology roadmap [8] as well as the text of several research reports generated by the PASS program.

2 Background

Flying has always been all about energy management. Common design and performance metrics that reflect this reality include lift vs drag, wet vs dry weight, endurance, range, fuel consumption vs altitude and pressure, etc. The change today from the time of the Wright brothers is that electronic systems—computation, communication, sensors, and actuators—participate in the tradeoffs in addition to issues of loft lines, structures, and weight. Several recent and proposed aero systems, shown in Figure 2, provide stark examples of energy management concerns and are discussed next.

Sky Tower. The renamed AeroVironment Helios will act as a communication relay in the sky. The goal is six months endurance using electric motors and solar energy for power. Not only must the system scavenge enough energy to sustain flight during the day, it must provide resources for night time. The Helios recently set an altitude record for its class when it just missed achieving a 100,000 ft. altitude goal.

Sensor Craft. This is an aircraft projected for the 2020 time frame with 70,000+ feet altitude, 80 hour endurance, and a sensor and computation suite that are barely imaginable today. Its skin will not only provide lift and structural support, it will embed a large antenna farm. Sensor Craft is the logical extension of today's Global Hawk with more than twice the endurance and an order of magnitude more capability. Counting ounces and ergs is essential to the ultimate success of the Sensor Craft vision.

Submunitions. These attack vehicles are parts of larger systems. They are typically battery powered and their missions are to use onboard sensors to acquire and identify targets then conduct lethal end games. Weight increases caused by sophisticated computers and sensors trade-off against range and endurance. The fact that these part aircraft/part bombs are single use mitigates strategies that would provide ample but more costly energy solutions.

Urban Flyers. The projection of increased engagements in urban environments has placed emphasis on small, agile, unmanned aircraft that can navigate through city streets and, perhaps, even in buildings. Whether flight is battery or petroleum powered, these vehicles must carry sophisticated sensors and communications devices and some may be only the



(a) Helios long endurance high flyer.



(b) Sensor craft for the future.



(c) Army BAT submunition.



(d) MARS flyer.

Figure 2: Flying vehicles with special energy needs.

size of a human fist. Energy management is a critical issue. An interesting variation on the theme is the hand-sized vehicles being designed for flight on Mars.

The military goal is always to maximize mission-performance metrics which factor in costs to perform, costs if not performed, possibilities of collateral damage, and targets of opportunity. The trade space investigated by the PACC program, in general, and the PASS program, in particular, is electronic power management in the service of mission goals. These tradeoffs may involve computation, communications, sensing, sensor exploitation, control surface actuation, and even use of electric engines. In some cases, the tradeoff is *how much* work to do given an energy constraint; in other cases, the tradeoff involves *how* to achieve the mission objectives. Two related examples highlight some of the issues involved:

1. A submunition has acquired a potential high-valued target but the current resolution of its location is insufficient to guarantee a successful attack. Resources are limited since it has taken quite a long time to acquire this target. One possibility is to attack a closer lower-valued target for which better location data is available onboard. This is a direct mission-value vs resource-availability tradeoff.
2. Three choices for the same scenario when the high-valued target will be pursued are (1) use the onboard sensor at higher resolution to resolve location ambiguity, (2) apply more sophisticated image-enhancement algorithms to data already onboard, or (3) use communications to share another platform's sensor data and fuse it with onboard data. Which of these options can we afford? And which one is the most effective use of our constrained energy budget? This is an example of a sensing vs computation vs communication tradeoff in the service of mission-goal achievement.

Resolving these technical tradeoffs not only presupposes sophisticated mathematics and performance models (e.g., sensor resolution as a function of energy), it also presupposes that we have models that accurately predict *mission performance in terms of the technical performance metrics*. Mission performance modeling is mostly beyond the scope of the PACC program so the PASS program must concentrate on those metrics that measure technical progress and intuitively support better mission accomplishment though we cannot always say by how much.

For example, algorithms that use less energy to achieve the same precision or voltage-scheduling technology that minimizes computation delay given a hard energy budget are surely valuable results and in scope. However, the steps that predict the increased percentage of target strikes per saved joule is beyond our means.

Thus, PASS research has concentrated on technology that will better manage and allocate constrained energy budgets. Our roadmap [8], on the other hand, takes a broader view and speculates about how technology of all types will come together to support more effective air vehicles for the military in the future.

3 Contractor Work Summary

The PASS program has supported three types of activities: (1) networking with other programs to share results, learn about key problems, and inform others of the PACC program, (2) completing CDRL items, and (3) research in core power-management problems important to avionics applications. Each is described in a separate subsection below.

Several research notes were written in the course of our investigations. One note [2] was published in a book on power-aware computing and two others [3, 7] have been submitted to refereed journals for possible publication. A fourth note [1] was published in the Morphware forum, an information exchange media sponsored by the DARPA IPTO Polymorphous Computing program. Appendix A includes the text of all research notes as well as the technology roadmap.

3.1 Networking with Other Programs

The PASS program home room is located in the NGC Integrated Systems (IS) Air Combat Systems (ACS). ACS is responsible for the design, development, and manufacturing of many aircraft for the U.S. military. The proximity to these programs provided an opportunity for many productive information exchanges. In particular, we conducted informal technical interchanges with the BAT, Smart Bombs, JSF, VTUAV, UCAR Watchkeeper, QSP, and Falcon program components within ACS. We informed them of the PACC program goals and ask them about their power-management concerns.

All of these programs, looking to their future needs, were concerned about weight, form factor, and packing density of electronics. The BAT and Smart Bombs programs have continually evaluated the tradeoff between powered and glide flight. The issue with the latter is the lack of a generator so batteries must provide all electric power. This in turn leads to a host of power-management tradeoffs though it can substantially reduce the costs of these one-use systems.

QSP and Falcon are concept design studies for future long-range hypersonic bombers and have a rather novel set of energy management issues. Skin temperatures can reach thousands of degrees Fahrenheit at the given flight speeds. The question is how to dissipate that heat. Further, computing, communication, and sensor systems and cockpits must be cooled. Though PACC technologies can substantially reduce the heat generated by electronics

and, hence, the cooling load, it doesn't address their major problems.

We have interacted with the Software Radio (JTRS) working group, one of the major PACC Phase II activities, and attend their semiannual working sessions held concurrently with PI meetings. We made a presentation [9] about the special problems of power-management to aircraft at their kickoff meeting.

3.2 CDRL Items

The PASS CDRL items include monthly financial reports, quarterly progress reports, a technology roadmap [8], and this final report. All CDRL items have been completed. The technology roadmap, in the form of annotated Power Point slides, is included in Appendix A. It is further described below.

3.2.1 Technology Roadmap

A three-step process was used to develop the technology roadmap: (1) assess ongoing engineering and scientific research, (2) determine which technologies were likely to mature in the next decade, and (3) project how those technologies might be packaged to have maximum impact. The goal of this activity was to take a broad look at where military aircraft technology was going; the investigation was not confined to power management. It was evident early in the investigation that power management was ubiquitous in the sense that it would increase the amount of onboard electronic capabilities available to support other technologies. So the more relevant issues were what the final products might look like.

The assessment of ongoing research was done by examining programs supported by DARPA, AFRL, ONR, and various national laboratories, as well as those initiated within academia. Openly available literature, web sites, and the informal technical interchange meetings mentioned above were the information sources consulted.

The key technology areas identified included computer hardware, JTRS and other communications advances, materials, human interface devices, intelligent systems, software development practices, sensors, miniaturization, security methods, design tools, model-based reasoning, and (of course) power management. The next issue was how advances in these areas might be combined and what the products might look like in the future.

The following technology packages are predicted for future aircraft based on the assessment and determination activities:

High-performance low-powered computer suites. Multi-chip motherboards will support multi-CPU chips and several will be packed in a shoe-box sized box. Multiple boxes will provide redundancy, and power dissipation will be less than 1 watt per GHz. Computing power of each box will easily exceed 50 GHz.

Process-centric computing models. Computing will be organized as a collection of processes, not subsystems. Individual processes will be movable among chips, motherboards, and shoe-boxes.

Onboard internet communications. The comm architecture will mimic the internet so that process locality, whether it be onboard or even remote, is not an essential parameter. That network will support connectivity with cockpits, actuators, sensors, and other systems distributed throughout the vehicle as well as offboard assets.

Multilevel secure computing base. Joint Vision 2010 and 2020 suppose an information-sharing structure that cannot exist without a supporting multilevel secure (MLS) infrastructure. Specialized process architectures will be developed to support MLS and the programmability of JTRS will be leveraged.

Autonomy within man/machine interactions. In addition to autonomous vehicles, the machine side of man/machine interactions will be very much more capable and provide primitive capabilities that rely on autonomous facilities for their enactment. For example, a simple request for better target ID might cause several autonomous rotorcraft to position, sense, and fuse target information.

Reconfigurable cockpit. Pilots will introduce themselves to an aircraft using a personal “ignition key” that contains their preferences for display arrangements and priorities as well as security keys and mission data. In addition, the keys will provide individual pilot preferences in control laws that effect the feel of flight.

Fusion-centric sensor suites. Today, sensors do an enormous amount of processing before sharing data. Often data transformations are performed that preclude the possibility of doing fusion at the correct level.

In the future, more raw data will be provided to the general-purpose computing suite where fusion with other onboard sensors and offboard information will occur at whatever level of representation is appropriate to current information needs.

Integrated health management. Health management will enjoy model-based technology and be more astute. Autonomic supply chain and ground-depot servicing support will be integrated with onboard health management. Ultimately, sortie production rates will increase while logistics and maintenance costs will decrease.

Formal design and requirements specifications. Methods to formally specify system requirements, architectures, and designs will be developed and used to support autocode generation and better verification and validation techniques.

Automatic code generation. Much of the system code will be generated from the above formal representations. For example, some aspects of flight-control software can be prototyped, in MATLABTM, from a symbolic mathematical representation. It should also be possible to autocode much of the process allocation, redundancy management, and onboard communication-scheduling software from formal specifications.

Formal methods. There will be methods to verify formal specifications of designs against formal statements of requirements. In addition, the same technology base will be able to support better methods of generating system test scenarios and improve the efficiency of verification and validation activities.

Subsystem procurement practice. In today's practice, procured subsystems are black boxes. The move to process-centric architectures and internet-style communications should change this as well as the way subsystems are procured. In the future, interface specifications will be protocol designs, mutually agreed upon by contractors and subcontractors and their work products will be a set of processes and protocols with dependencies developed *during the detailed design period*, rather than before the subcontract is signed.

Smart skins. Aircraft skins will embed new miniature electronic devices such as sensors and antennas and MEMS devices will provide the ability

to dynamically reshape the skins to effect aero parameters and signatures. Skins may also have embedded robust communication fabrics which form onboard networks.

Form-fit packaging. The inside of an aircraft is an irregular shape and almost all of the packages that go there are rectangular. New light-weight materials can be economically molded to specialized shapes (directly from CAD specifications) to make better use of the available space and reduce weight.

3.3 Research Investigations

This section discusses PASS research investigations in application-level reasoning, computation vs communication, redundancy management, and energy and time management. Each topic is addressed in a separate subsection.

3.3.1 Application-Level Reasoning

If the ultimate goal is to apply technology to enhance mission value, there must be methodology to inform the system about the mission-value tradeoff space. Left to their own devices, technology providers can only supply “less is better” improvements, e.g., minimize the expected energy used to execute an algorithm given a hard realtime deadline. However, real power-management problems are more systemic.

Consider a submunition that has acquired a target that it expects to strike in 45 seconds. The remaining energy store is limited so the relevant question/optimization is the best allocation of that store among sensing, actuators, and computation to improve the probability of a lethal strike. No abstract technology can solve this problem without knowledge of performance as a function of allocation and a map from performance to mission value.

Two PASS studies were devoted to this problem class. The first [1] developed an API (application program interface) with the expressiveness for applications to share knowledge with the underlying technology infrastructure in order to jointly optimize system behavior. The key information presented at the interface included alternative computation modes, an organized mission model, and metrics that described complexity, priorities, and costs. The concentration was more on what information needed to be shared than how to use that information.

The second study [2] investigated energy usage vs system performance. Two problems were formulated and solved. The first dealt with the effect of energy efficiency on aircraft endurance. The linkage was that (1) energy efficiency reduces the weight of the battery or generator, (2) weight reduction allows more fuel, and (3) extra fuel increases endurance. The second problem investigation postulated that a sensor's ability to reduce variance is a function of the energy that the sensor can use. The particular problem addressed assumed a total energy budget and two sensors whose output would be fused to further reduce variance. The goal was to find the optimal energy allocation between the two sensors to minimize residual variance. Both problems were formulated with simplified models so that the problem spaces could be explored analytically.

3.3.2 Computation vs Communication

Computation vs communication tradeoffs exist in military aircraft applications, unattended ground sensor nets, space, and other domains. For example, should available data be processed locally to reduce its size before transmitting it or should the raw data be transmitted? Since both computation and communication consume varying quantities of energy, this is an important power management problem.

We performed a study [3] in this area that addressed a core theoretical problem: determine the class of functions that have associated local algorithms that substantially reduce bandwidth. Not all functions belong to this class. Computation of the ordinary median, for example, does not admit bandwidth-reducing local computations so transmitting raw data samples to another site is as efficient as any other implementation.

We formulated a conjecture about necessary and sufficient conditions that a distributed computation could reduce bandwidth and proved the condition sufficient. However, necessity is still an open problem though we proved it for the class of monotonic functions. The conjecture involves the existence of a bicontinuous map from the function's domain to a range that is isomorphic to the equivalence classes induced by the function's values.

3.3.3 Redundancy Management

Redundant process executions on separate hosts is a popular technique to increase system robustness and reliability. However, a straightforward im-

plementation of n -way redundancy consumes n times as much energy as non-redundant computation. We conducted an investigation to determine if it is possible to do better than “straightforward.” Note, hard realtime deadlines were assumed for the computations. Two techniques were investigated.

The first [4] considered how to implement a triply redundant computation to minimize expected energy expenditure. The strategy was to execute two of the three processes relatively rapidly. If the two processes agreed on their results, all computation ceased. If they did not agree, the third process, executing initially at a slower rate, was sped up to complete its computations by the deadline and vote its results. The energy consumed per computation cycle was a function of processor speed, a controllable variable. The optimal processor speeds and energy consumed were computed as a function of the probability that the two-process rendezvous was unsuccessful. The relative savings were found to vary monotonically from 1/3 to 0 as a function of failure probability—not a surprising result. The math models determined the best execution speeds for all process segments and showed that, when the failure probability was 1/3—a very high number—then all three processes should execute at the same speed. When the probability is lower, staggering execution speeds was a winner by an amount dependent on the failure probability.

The second study [5] considered how to implement duplex redundancy to minimize expected energy consumption. The model used is that the same process chunk was executed on two processors and results to-date compared. If they agreed, the next chunk was executed; if they disagreed, both processors rolled back and re-executed the previous chunk, etc. The full problem statement assumes that the probability of agreement is known and that there is a hard realtime deadline. When there is a disagreement, the execution must be sped up since more computation must be done before the deadline. The expected number of rollbacks and the incurred penalties are functions of the chunk *size* as well as the probability of agreement. So the problem solved was to determine the chunk size that minimized expected energy consumption and still met the deadline. The process and energy models used therein are described in [6].

3.3.4 Energy and Time Management

Many modern computer architectures provide control knobs to regulate energy expenditure. Executing programs can change the supply voltage and

the clock frequency. When voltage is increased, the CPU can sustain higher execution rates so the clock frequency can be increased though there is a higher per-cycle energy cost. The class of applications investigated by PASS typically have realtime deadlines as well as energy management needs so these controls present an interesting opportunity. We explored a broad set of related problems using several process and power models and various optimization criteria [7].

Two process models were investigated. The first, the simple probabilistic model, provides a function p , where $p(x)$ is the probability that process complexity is exactly x cycles. The second model, the structural model, posits that a process is a set of simple segments. At the end of each segment, there is a multi-way branch to other segments. The probabilities of these branches are given along with the complexity (cycle count) of the segments. Conditional terminals are those segments where the sum of the outgoing branch probabilities is less than one; if that sum is zero, the segment is a simple terminal.

The class of power models examined was a two-parameter family. If Π_{mn} is a power model, then the energy expended to execute c cycles is proportional to cv^m , where v is voltage, and the time to execute c cycles is proportional to c/v^n . Aside from the fact that threshold voltage is ignored, this is a reasonably realistic set of power models.

Two optimization problems were solved for both process and all power models. The first asks for the voltage schedule that minimizes expected execution time given a fixed energy budget. The second problem asks for the voltage schedule that minimizes expected energy utilization given a hard realtime deadline. The first optimization seems to be novel while the second has an extensive literature. A voltage schedule is simply a specification of the voltage to use when each cycle is executed. Another problem was formulated and solved for the simple probabilistic model: Let $Q(E, T)$ be a penalty function of total energy, E , and total time, T , taken to execute a process; Find the voltage schedule that minimizes the expected value of Q .

The most interesting result of this study was an insight that determines when voltage level, hence, rates of energy expenditure and computation should change and when it is optimal to remain constant. Voltage will only change in optimal solutions of structurally modeled processes when something new is learned about future process complexity, and that can only happen when a branch is selected at the end of a simple segment execution. Voltage will only change in optimal solutions of simple probabilistic

models when something is learned about future process complexity, and that can only happen when executing a cycle where the probability of termination is not zero. What is learned is that *the process didn't terminate*. This information-theoretic insight is robust across all power models.

4 Lessons Learned

Our research studies provide point solutions for many energy and time management problems and these results are available to incorporate in future systems. However, our key insight was developed through investigations and interactions with other PACC programs:

Power-aware computing is a systemic problem that involves application and mission expertise as well as the base technology infrastructure.

While less-is-better technology will certainly improve corporate enterprise solutions and mobile computing, more is needed for military systems. Application value systems, based on mission analysis and objectives, must be used to determine (1) links between performance and mission metrics and (2) choices among the set of possible mission realizations.

In other words, application knowledge must be linked with supporting methods to manage time, energy, and other resources to optimize missions. That linkage must occur during mission execution as well as in the software-engineering laboratory. During the mission, the application systems must track its own state and the mission to determine what is essential and what is not. The resource-management layer must use this information to control computation, communications, and sensing. However, the application must be able to reorganize what it is doing and how it is doing it to assure success when resources are limited.

The software-engineering laboratory must be used to measure the complexities and costs of alternative computations and establish the control knobs that are adjusted by both the application and the resource management layers. Further, the system structure must be made overt as to mission phases and computational modes so that the complexity information can be properly applied and future system needs can be anticipated at runtime.

Support of this vision surely needs new technology, some of which is being developed by the PACC program. But new *development processes* are needed too. Further, there must be ways to develop large chunks of applications with little regard for low-level details of power management. Otherwise, the resulting code will be too brittle and too resistant to refresh and modification. The PACC program is making progress in all of these areas, but much more remains to be done by future research efforts.

A Appendix: Research Notes

The listed research notes comprise this appendix and follow in the enumerated order.

References

- [1] J. A. Barnett, Intelligent reconfigurable systems: The ubiquitous API, in *PCA Morphware Forum*.
- [2] J. A. Barnett, Application-level power awareness, in R. Melham and R. Graybill (eds.), *Power Aware Computing*, Kluwer, 2003.
- [3] J. A. Barnett, Distributed computation: A conjecture, submitted for publication.
- [4] J. A. Barnett, Energy efficient redundancy, NGC research note.
- [5] J. A. Barnett, Power-aware duplex redundancy, NGC research note.
- [6] J. A. Barnett, Minimum energy process execution, NGC research note.
- [7] J. A. Barnett, Dynamic voltage scheduling optimizations, submitted for publication.
- [8] J. A. Barnett, Avionics for the 21st century, NGC research note.
- [9] J. A. Barnett, Power-Aware challenges for unmanned air vehicles, presented to PACC Software Radio working group.

Intelligent Reconfigurable Systems: The Ubiquitous API

Jeffrey A. Barnett
jbarnett@nrtc.northrop.com
Northrop Grumman Corporation

August 2002

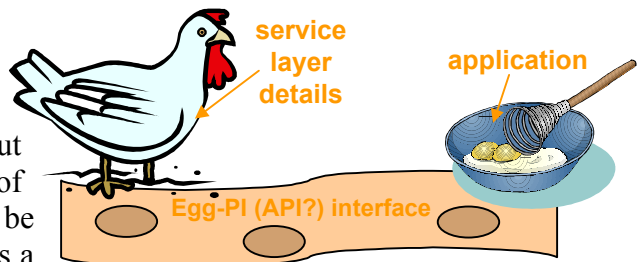
ABSTRACT

Modern software engineering and programming practices feature system-organization approaches that assist development, simplify maintenance, and anticipate refresh. The key concepts therein are abstraction layering architectures to organize system designs, and application programming interfaces (API) to isolate application-specific code from the supporting hardware and runtime infrastructure called middleware. From the point of view of the software engineer, many details such as configuration, hardware status, and time and resource management are swept under the rug. Unfortunately, some of that detail is properly part of an application's problem-solving domain in areas such as realtime systems or use of exotic hardware. The conflict is reviewed and an approach—the ubiquitous API—that addresses the inherent issues is suggested so that hygienic software engineering technology will be available in these domains too. The idea is an API that lives in the development laboratory as well as the execution environment. If the recommended technology were realized, intelligent systems that could reconfigure themselves in response to failures and resource considerations would be available.

Keywords: Software engineering, API, realtime, power-aware computing and communications, polymorphic computer architectures, reconfigurable smart systems

1 Overview and Summary

A half-century of software engineering has produced results and there is consensus about fundamental approaches. The main point of technical consensus is that a system ought to be a collection of layers, where each layer offers a well-defined, stable functional interface that supports the other layers built upon it. In fact there is a consensus about the sort of interfaces that ought to be used for components such as operating systems, database servers, and other middleware products. That consensus is not about the specifics of interface functionality. Rather it is about what sorts of issues should be exposed in a middleware layer's interface and what should be encapsulated by its implementation. Typically, the encapsulation hides details of hardware configuration, device timing, algorithmic specifics, communications media, and the like.



The consensus layering solution works well for most systems. Requirement specifications are shorter, design times are reduced, programmers are more efficient, and technology refresh is simplified. However, there are important classes of systems that are not compatible with the standard solution. Realtime systems are such an exception. The issue is that matters of timing, configuration, error recovery, and others normally encapsulated by the middleware are legitimate application-level concerns. Since the necessary mecha-

nisms and control knobs are hidden in the lower level implementations, realtime programmers must either create a kluge or implement their own version of the middleware. Both possibilities have negative impacts on system costs and lifecycle timelines [4].

Current hardware architecture research points to future system organization strategies that will also need to break the standard middleware layer interfaces. The power-aware computing and communications (PACC [5]) program will provide computer bases that can regulate their energy consumption profiles in response to circumstances. When energy is scarce, the hardware can ramp down its rate of expenditure and the software can do less or do different e.g., adjust the mission goals. In times of plenty, system goals can be pursued more aggressively. The polymorphous computer architecture (PCA [6]) program will provide hardware bases that can change their fundamental architectures, e.g., FIFO versus RAM or hypercube versus streaming, to optimize total system performance. Both of these DARPA programs require that application layers cooperate with the middleware that controls the novel hardware. Timing issues, device modes, and low-level configuration details must be jointly optimized by the system as a whole. As with realtime systems, the standard middleware layers hide the wrong information.

Some observations about where the knowledge resides to properly structure these systems are in order. Realtime system requirements and designs determine execution timelines, redundancy models, and error recovery strategies. The designer of a power-aware system is the one who understands, for a given circumstance, whether it makes more sense to communicate less or process sensor data to a lesser precision. The architect of a large numerical code is the one who best appreciates whether a part of the system is better served by an array model or a pipeline. In all of these cases, the key insights about how to deal with low-level management tasks at execution time are based on knowledge derived in the software-engineering laboratory. This conclusion suggests a solution to the system organization problem caused by standard layering technology and assumptions. I call that solution the *ubiquitous API*.

Sections 2-5 elaborate the discussion to this point and Sections 6-8 introduce the ubiquitous API and review related research. An application-programming interface (API) is the functional interface to a middleware layer. Thus, an API is a set of calls and interactions between an application and the middleware at execution time. A ubiquitous API has a presence in the development laboratory where vital information resides as well as the runtime environment. Application specifications such as problem-solving modes and configurations, source code, event definitions, and execution analyses are provided to a special compiler where they are combined with resource descriptors and middleware primitives. The compiler output includes an extended API and precompiled strategies to react to dynamic events in addition to the application binaries. The runtime is now properly equipped to respond efficiently to total system needs and provide meaningful information exchanges between the application layers and the supporting infrastructure.

The discussion of the ubiquitous API also provides a solution for how very complicated information, e.g., application modalities and configuration strategies, can be encoded for the compiler. The suggested representation is a state machine where the states are con-

figurations (basically software architectures) and the state transitions are triggered by events (application signals such as CHANGED MISSION TIME or infrastructure signals such as BATTERY LOW). Transition specifications include information that relates components of the “from” and “to” states so that proper system state can be maintained. State machine representations will make both development and analysis relatively straightforward.

The ubiquitous API preserves the software-engineering concepts of layering and separation of concerns as much as is possible. However, it has methods to determine who should be concerned—the application, the middleware, or both. When the answer is both, it provides the tools necessary to integrate a total system cleanly even when the inter-layer co-dependencies are quite complex. As a result, systems can intelligently reconfigure themselves in response to events and resource considerations.

2 Modern Software-Engineering Approaches

Dave Parnus opined, in the early 1970’s [9], that the criterion for good software modularization is that each module should hide a related set of implementation decisions. His observation was based on some early versions of object-oriented (OO) languages such as SIMULA and common sense. Computer systems could be developed that functioned better, cost less, and were easier to maintain if all inter-module dependencies are explicit and occur only at the boundaries between modules instead of relying on internal implementation details. The reason was simple: modules could be repaired or improved without effects on the using modules as long as the interfaces remained constant.

OO languages typically provide encapsulation and functional interfaces that specify the types of arguments expected, values returned, and whether there are side effects or not.¹ Unfortunately, early OO tools were slow, buggy, and not widely available. They were also rather limited compared to more modern OO tools because of resource limitations and lack of experience with the concepts. As a result, most systems built with these facilities tended to be a simple stack of software layers. Each layer provided an abstraction (AKA a virtual machine) on which the next higher layer was implemented. For example, an OO trip-planning system might be comprised of the three layers shown in Figure 1

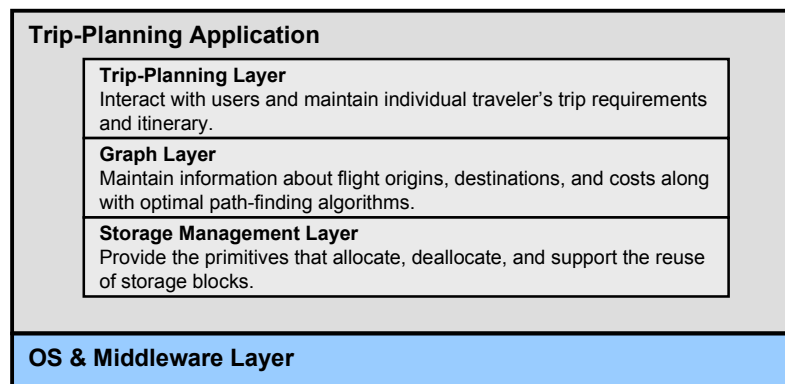


Figure 1: Layered Trip-Planning Application

¹ A third feature, inheritance, probably the most interesting and powerful aspect of an OO approach, is not germane to the present discussion so will be ignored here. The representation described in Section 7, however, could use inheritance to simplify specifications and application rehosting.

where the trip-planning layer is built on the graph layer, which in turn is built on the storage-management layer. The operating system (OS) and associated middleware support the whole application stack here as they do in most applications.

An interesting observation is that the advantages of OO—encapsulation to hide implementation decisions and functional interfaces—are available without OO facilities. A decent design and some coding discipline will do the job nicely. In fact that is today's most popular architectural strategy: layered designs along with detailed interface specifications. The strategy works very well as long as the discipline is there; Good discipline often will marginalize the value of OO tools.

The bottom layer of most software systems is the OS. It provides access to hardware execution engines, resources such as file storage, and protection against accidental or malicious damage by other software or agents. In the early years of the modern era, up to the mid 1970's, most operating systems were small collections of device drivers, resource allocators, and a scheduler. Each machine type had a different OS and the interface between it and the applications was ad hoc. That interface often varied significantly between different models of the same computer family. As the role of the OS became more substantial with the advent of networking, multimedia applications code, requirements for backups, and sophisticated user interaction support, the ad hoc interfaces became an expensive nuisance.

The approach that evolved in the OS arena is similar to that used for application development. The OS was simply viewed as a software layer with a well-defined interface between it and the applications that it supports. This application programming interface, or the API as it is commonly called, is now prevalent for almost all frequently used services. API definitions now exist for families of database services, information exchange, networks, and high-performance computer centers in addition to standard OS services. Some API specifications, e.g., POSIX, CORBA, and SQL, are now maintained and nurtured by national and international standards organizations. The result of this move, along with standardization of many popular programming languages, is an ability to develop applications that can be hosted on a large number of different computer types and can use services from a variety of vendors with little preplanning or reprogramming. The term *middleware* is frequently used to denote operating systems as well as other infrastructure that supports multiple applications. The cost of developing middleware has been substantially reduced too. So the world is good.

3 Hidden Details

The use of a formalized API for an operating system sweeps many important implementation details under the rug. While so located, these details are removed from the application's purview and control. That of course is the point of a good API. Table 1 lists several of the concerns that middleware, particularly an OS layer, encapsulates. The breadth of those concerns is rather impressive and continues to grow. Some old timers fondly remember the joy of reinventing approaches to all or most of these problems for each new system that was built. Our paymasters are much happier today because the modern approach reduces cost as well as enabling development of more capable systems.

Table 1: Details Swept Under the Rug By OS & Middleware

Detail Category	Hidden Details
Processor Features	How is the floating-point unit connected? What kind of caches and how large are they? Is this a single processor or multiprocessor main board? How wide is the pipeline? Can it be reconfigured?
Hardware Devices	What devices are attached? At what rate do they operate? What are their interrupt priorities? What are the handshake time limits? What type of power supply is used? Can it report its status? Are printers spooled?
Configuration	Which computers are local? Are files shared through a common server or is FTP used? Where am I executing? What else is on this node? Is the execution redundant? Where are the service providers?
Error Recovery	Did <i>that</i> disk fail? If so, where are the files now? Is there a checkpoint for rollback? Has any redundant execution stayed alive? Is there a communications path from here to there? What is its timing?
Deadline Management	How long (wall clock) will this computation take? How long (CPU time) will this computation take? Which process needs to execute next? What resources need to be dedicated? When is this process' output needed by another process?
Physical File System	Where are my files? How are the indices calculated and stored? What is its layout? Is another process accessing <i>this</i> file?
Resource Policy	Is the distribution of goodies fair? What is best for the system? How much disk usage is reasonable? Do file pages and virtual pages count the same toward the budgets? What is the scheduling algorithm?
Service Location	Where is the database? Which timeserver should we use? What is the load on each SQL server?
Media & Protocols	What is the speed of the LAN? Should the LAN or the WAN be used? Is a link or packet connection more sensible? Am I adding to the congestion? Should the back-off and retransmit policies be adjusted?
User Resources	Does the same box support the user and his computations? Should the graphics hardware accelerator be used or is the library code more efficient? Should user keystrokes be given priority over background processing? Is it time to password lock the terminal?

4 Specialty Domains & Resources

This section considers questions about the sort of details that a good layer will encapsulate and hide from its users. The answers will be used to investigate appropriate layering for the class of realtime embedded systems. Since proper allocations of resources, error recovery, and dynamic configuration are first-class application concerns in this domain, we will see that the conventional wisdom of what should be encapsulated by the API and what is inappropriate do not apply here.

Two active DARPA programs—Power-Aware Computing and Communications and Polymorphous Computing Architectures—also raise fundamental questions about an API and what it can properly encapsulate. Both programs are developing computational infrastructure that has unique potential to respond to application value systems about what is important and, thus, how resources should be configured and allocated. As with realtime systems, conventional methods are at odds with application needs.

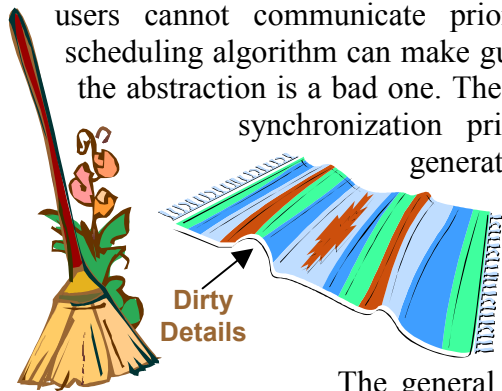
Sections 5-7 discuss how one could design an API family that preserves the benefits of modern system organization concepts while serving specialty domains and using the new resource management approaches that will be made possible by current research.

4.1 What a Good Layer Hides

The Parnus dictum that a module ought to hide a related set of implementation decisions has served us well for many decades. However, the discussions in Sections 4.2 and 4.3 will indicate that the following emphasis is necessary:

A module ought to hide a related set of implementation decisions that are not of *proper interest* to the modules that will use it.

Consider an application that requires a particular process to have priority execution privileges when not all can be served in a timely manner. A scheduling layer where the users cannot communicate priorities is inadequate. Even if the embedded scheduling algorithm can make guarantees, e.g., maximum or fair use of the CPU, the abstraction is a bad one. The application would need to make frequent use of synchronization primitives to exchange status information and generate its own priority implementation. In other words, the application would need to reinvent priority-scheduling technology. Experience has shown this move often will lead to unexpected problems [2].



The general principle is that if a layer is responsible for a functional or performance requirement then either (1) that layer must satisfy it on its own or (2) lower layers must provide, at their interfaces, adequate guarantees to entail that the requirement can be satisfied. In other words, not everything can be swept under the rug and what must be exposed depends on the specific application.

4.2 Realtime Embedded Systems

Realtime embedded systems are becoming more prevalent in the modern world. They not only perform traditional roles in environment control, avionics, and manufacturing; today, these systems populate appliances, home computers, and portable entertainment devices [7]. However, this discussion will consider only a single traditional realtime applications area, avionics. The avionics domain requirements and features pretty much subsume those of other examples. The reason for the investigation is to understand which issues are first-class application development and maintenance considerations. From this discussion, we will be better able to decide what details can be properly encapsulated by middleware. That decision has two important entailments: (1) whether COTS products can play a meaningful role in the domain and (2) how these systems can be layered.

Table 2 summarizes some of the concerns that are part of avionics applications but are typically dealt with by the middleware in the non-realtime world. The way to best appreciate the significance of this table is to compare it to Table 1. The conclusion is straightforward: virtually every issue swept under the rug in the layering of ordinary applications is a first-class application concern for realtime systems.

Many details that are resolved at runtime by standard middleware layers are, in fact, *resolved at design and development time* in the realtime arena. This means that realtime

system designs and code are replete with the sort of detail that many decades of software engineering have attempted to take out of the hands of application programmers. As a result, realtime systems remain as some of the most costly ones in the modern world. The cost and man-time per line of code/module are higher than in any other domain [4].

Table 2: Avionics Realtime Application-Level Concerns

Area	Description
Redundancy	Specific redundancy strategy, e.g., n-way voting or hot backup, is part of system design/requirements on a subsystem-by-subsystem basis.
Error Detection	Error detection is a substantial application activity and will involve data semantics as well as obligation failures. Code incorporates self-diagnostics (BIT) and continuation methods from self or other errors.
Error Recovery	Recovery is often by preplanned reconfiguration or by reinitializing subsystems. Alternate computation paths are used during recovery.
Resource Management	All critical tasks are guaranteed adequate resources. Thus, exact consumption must be known preflight. This includes pre-allocation of the electric power budgets, CPU cycles, RAM, and bandwidth.
Coordination	Computation pipelines originating at devices, passing through subsystems, and terminating in device controllers or output actions are a frequently occurring paradigm.
Timing	Both ends of the pipeline may have precise, periodic timing requirements. Therefore, processes and communications must be co-scheduled to meet these requirements.
Device Management	Specific devices must be connected to specific computers because timing and error recovery mechanisms assume this knowledge.

4.3 New Computation Bases

Several current research programs are innovating new ways to organize computer and communication resources in order to provide more useful problem-solving power. Two of these programs are discussed in this section—Power Aware Computing and Communication and Polymorphous Computing Architectures. Both are sponsored by DARPA and both will stress the decision about what details can be swept under the middleware rug and which ones cannot.

4.3.1 Power Aware Computing and Communications (PACC)

The PACC program [5] is based on two premises: (1) equal results from less energy investment is always a good thing and (2) applications can achieve more functionality if they adapt their behavior to resource availability. The hardware that is being developed provides dynamic power versus performance adjustments, e.g., the issue-width of a pipeline, methods to turn off unused units and quickly revitalize them when needed, voltage adjustments, and adaptations to observed computational patterns. In addition, specialized middleware components including compilers, schedulers, and libraries are being developed to support integrated power-aware applications [8].

These technologies can, indeed, produce equal results for less energy investment. However, they do not fully support intelligent applications, those that understand what is more and what is less important, so that mission-specific energy utilization tradeoffs can be made. Consider a smart submunition, e.g., the BAT [10], without a motor or generator, powered by a battery with limited capacity. A nominal mission is, say, three minutes

counting from the moment that the submunition is dispensed. The battery must be used to power an IR and an acoustic sensor, computers, and flight actuators until the submunition makes a precision strike on its target.²

Aggressive energy consumption provides many benefits: (1) more frequent sensor scans, (2) utilization of better fusion algorithms and trajectory computations, and (3) more exact flight controls. If the submunition acquires a target that is very close, say one minute away, these benefits are available because there is more energy per unit of remaining mission time. On the other hand, if the target is distant, energy must be husbanded. In this example, the sensor management strategy, choice of computational algorithms, and flight control laws must vary depending on the situation; and the application is the sole judge of what the impact of a situation should be [1].

Only the application can switch algorithms and only the lower system levels can control the hardware base and measure things like remaining battery energy. In other words, the application and the supporting infrastructure must cooperate in configuring the system for optimal achievement of the total mission. This is simply not possible if all the energy controls (or even knowledge of the existence of these controls) are encapsulated in the middleware and the middleware doesn't *understand* the application's value system. (Is it better to sacrifice sensor scan rate, in this situation, or guidance precision?) Thus, the intelligent use of power-aware technology depends on restructuring the system layers and the API's that glue them together. It is interesting to note that the most stressful applications of PACC technology will surely be in the realtime arena.

4.3.2 Polymorphous Computing Architectures (PCA)

Different sorts of large computations can benefit from fundamentally different types of hardware architectures. In the past, developers of massive computations studied the available hardware then attempted to tune their code to that hardware [13]. This has never been a particularly effective way to develop systems and the PCA program [6] is looking to provide better paradigms. The essential idea is that application developers assume an appropriate, perhaps optimal, architecture is available and the hardware will make the necessary adjustments, e.g., configure itself as a hyper-cube for one application and as a stream machine for another. In fact, given that reconfiguration is fast enough, morphs are possible even in the middle of a single application. The pain in developing large numeric codes will clearly decrease if PCA is successful.

The researchers are developing many interesting approaches to morphing: adjustable memory width, whether it is a cache, RAM, or FIFO, key size, and number of ports; re-organizable communications mechanisms and flexible CPU cores that can be configured as pipelines, in loosely coupled configurations, or in various asymmetric patterns; etc. The possibilities are quite expansive and the idea is rather fascinating. However, there are fundamental questions of how to use this technology: How does an application developer

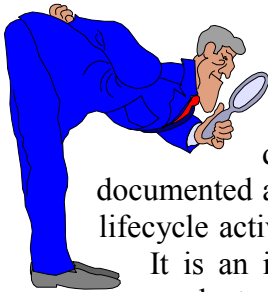
² Many current submunitions use thermal batteries so that there is a fixed amount of energy available per unit time. Energy that is not used is wasted and requirements above that fixed amount cannot be satisfied. For this example, assume the use of more normal battery technology. Future designs of smart submunitions may include engines and generators raising different sets of issues for power-aware implementations.

determine what is an optimal architecture? How is this information communicated to the middleware that is responsible for the configuration? How is dynamic control exercised while a computation changes modes? Is it the application's responsibility or is it the middleware that must make optimization decisions? What is optimized if more than one application shares a morphing configuration? Are the competing applications responsible to work and play well with one another? Does security restrict resource-sharing policies?

Reference to Table 1 shows, as it does for PACC applications, that standard system layering is inappropriate for PCA. Details about processor features, hardware devices, configurations, resource policies, and media and LAN protocols are a proper interest of the application programmer as well as the layers comprising the middleware. Once again, the wrong details have been swept under the rug.

5 Who Knows What and When Did They Know It?

Realtime systems place stringent and exacting requirements on configuration, timing, and resource allocation policies. Power-aware systems must dynamically reason about which behaviors, including alternative algorithms, are worth the investment of resources and which are not. Developers of computations that execute on polymorphous hardware must be keenly aware of the interactions of their algorithms with hardware architectures. The common thread in these examples is that application programmers must consider the sort of low-level details that are usually encapsulated by middleware and removed from application-level control. In other words, the API is too narrow and too ill-informed for the middleware to properly support the system types cited here.



The individuals who best understand an application's value system, requirements, and its preferred inter-layer interactions are its designers and programmers. Configuration, timing, and architecture decisions are made and anticipated when the requirements are documented and the software design is elaborated. In many instances, these early lifecycle activities determine which hardware and middleware will be purchased. It is an illusion that requirements and software design are general-purpose products. In the specialty domains discussed herein, the whole development activity is an exercise in integrated optimization. Even the dynamic needs of a power-aware system must be anticipated and the supporting infrastructure—application level and other—must be provided in the software-engineering laboratory.

Another issue is how a system develops a response to changes. What specifically does a realtime system do when the B bus fails? Does a power-aware application sacrifice sensor rate or communications to increase mission time when its target moves away from it? Is it worthwhile to morph the hardware to better service *this* relatively short computation vignette? My guess is that developing proper answers to these questions involves serious computation, simulation, and testing in addition to designer intuition. In other words, neither appropriate answers nor devising methods to implement them is possible at runtime. Of course that is the only available option if all interactions between the application and supporting infrastructure must take place at runtime through a standard API. There is a better approach and that is the topic of Sections 6-7.

6 The Ubiquitous API

Much of what we have learned from the previous sections of this document is summarized Figure 2. This section attempts to elucidate the resulting insights by proposing a novel system development and execution architecture; the next section provides technical details about the descriptor abstractions that are necessary. The approach entailed by the

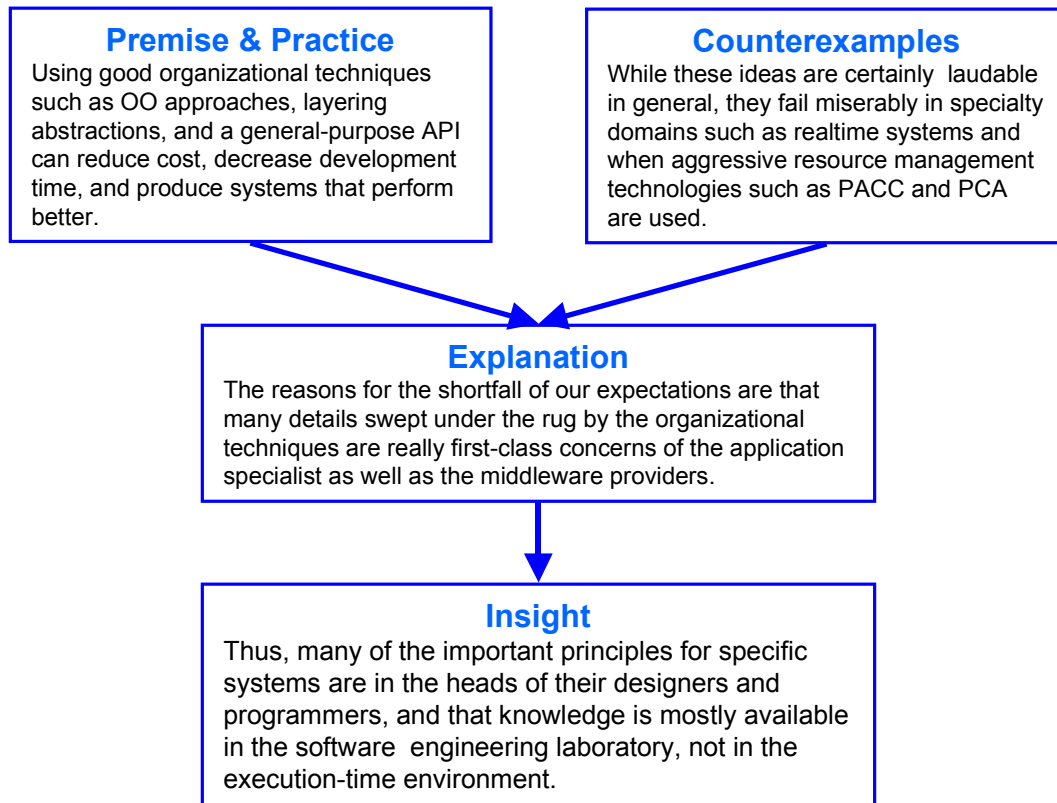
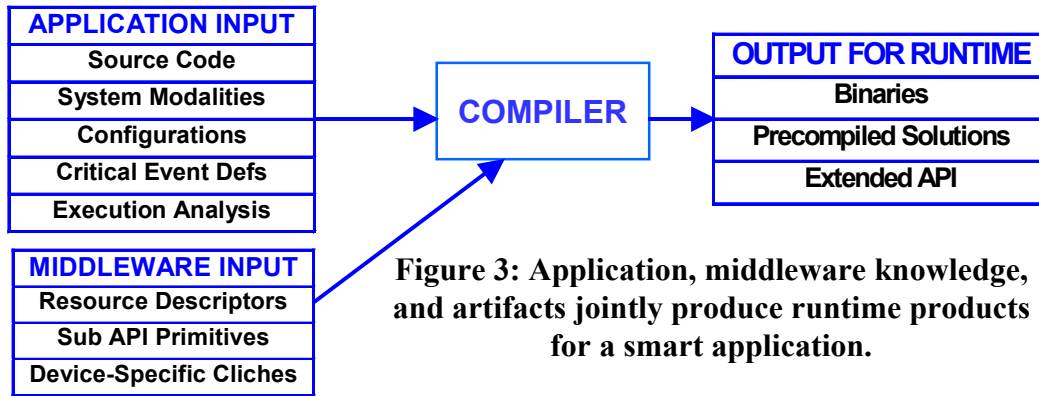


Figure 2: Summary of the argument in Sections 2-5

argument in the figure is straightforward; *the system layers must interact in the software-engineering laboratory at development time*. A traditional API is good for communicating a few numerical parameters and strings, e.g., file handles and path names. That mechanism, however, is singularly ill-suited for the transmission of more complicated data such as structural information, e.g., systems configurations, preferred dynamic response to events, e.g., power-management strategies as responses to mission timeline changes, and the like. The traditional API is also ill-equipped to carry on dialogues between the application layers and the middleware to do mutual problem solving for the good of the system. Both can make valuable contributions. The application knows best what is important and what it can do differently [1]. The middleware knows best how to implement changes and tune the system resources and configuration.

The solution that I envision has two parts: (1) an extended development environment where the application exposes its structure and built-in assumptions and (2) an execution environment where the knowledge gained at development time is put to good use. Figure 3 shows the development environment where the application and its descriptors are combined with middleware how-to knowledge and resource characterizations by a special



compiler. The output of the process is used to build a runtime environment optimized for the application.

The application obviously provides the source code. That code will probably be laced with calls on generic API primitives as well as pragmas that provide hints and clues about structure, performance, and assumptions. Other information that must be provided by the application concerns the range of computations that can/will be performed. That information comes in two forms: (1) modalities that describe individual problem-solving epochs such as preflight checkout, attack mode, or large matrix inversion and (2) configuration descriptions that declare active subsets of the software, and coordination methods, to support the various modes.

The set of modes will vary substantially from application to application and will also reflect design-time assumptions such as whether power management is in play, the hardware is morphable, or error management and real-time deadlines are domain concerns. The application must also declare the sort of events that are detectable by itself or the middleware that could trigger mode or configuration changes. The executing system must have methods to exchange information about occurrences of these events in order to induce cooperative reconfigurations.

Finally, performance and resource consumption estimates are necessary in order to manage the system. If one doesn't know the energy consumption statistics of two different, alternative configurations, how can a choice between them be made in order to conserve energy? Similar performance information is needed to manage realtime systems when there are resource problems due to component failures and/or work overload as it is needed to make intelligent choices about whether or when to morph the hardware base. There is clearly a need for simulation and benchmark facilities (not shown in the figure) in the software-engineering laboratory. It is also possible to capture some of the necessary data during system execution and provide feedback to the development tools.

The compiler shown in the figure is actually a collection of analysis and generation tools that are responsible for accumulating and communicating sufficient information to the runtime environment to manage application executions. The middleware layers contribute part of the compiler input. If hardware resources are to be managed, then descriptors of their configuration and the available control knobs are needed, as are special library

functions and macros, to support device-specific computation cliches such as streaming, array-processing, or low-energy mode. In addition, libraries of subprimitives and their descriptors are needed in order to build (compile) an appropriate API for *this* application.

The output of the compiler is more than the binaries produced from the source code. It must include an extended API that provides for bi-directional exchange of event triggers, mode switch notifications, detected anomalies, and parameters that measure mission life-time, energy availability, performance estimates, and more. The compiler before runtime determines the nature and extent of this extended interface. In addition, the compiler will develop sets of predetermined solutions, perhaps in the form of scripts that guide rapid reconfigurations of the hardware and software in response to events. These solutions may be parameterized. Consider an energy-saving mode where two adjustments are necessary: The first is the instantiation of a more frugal algorithm and the second is reducing CPU speed to save energy. The CPU speed selection will depend on (1) the deadline for the computation, (2) the remaining battery resources, (3) projected needs of future computations, and (4) the settings that this CPU can implement. All of this information and more, e.g., the availability of another execution site and the cost to move there, could be used to parameterize a configuration change or decide it is not worthwhile.

As is readily apparent, a great deal of an API and its implementation are determined in the development environment. In fact many of the interactions that would otherwise be needed at runtime are now handled up front. Since the API would reside in both the development and the execution environments, I will call this approach the *ubiquitous API*. Its manifestation in the development environment is the compiler.

The execution system is a simple layered product as shown by the inset. The application sits on top of the extended API used to convey information between the application and the middleware as well as providing the normal types of command structures. The middleware interprets the cache of precompiled strategies to determine its reaction to API calls and events. The drivers are used to command the hardware and resources as well as detect low-level events. Events from these sources also cause the middleware to consult the precompiled strategies to determine its reaction and inform the application. The magic in this approach is not only how well the system can serve application's requirements, it's also in how efficient it can be. Typically, efficiency is a product of investment. Here, a large part of that payment is made at development time where it is affordable instead of at execution time where finding solutions would compete for the same resources that are being managed.

Application Code	
Extended API	
Middleware	
Drivers	Strategies
Hardware & Resources	

Consider the inset figure as a composite view of the technology for both the development and the runtime environments. It is a simple three-layer view of a total system. The application is built on the ubiquitous API that in turn is built on top of the supporting infrastructure consisting of hardware resources and standard middleware components such as an operating system. Clearly the supporting infrastructure will con-

Application System
The Ubiquitous API
Supporting Infrastructure

tinue to evolve as long as better hardware, resources, and middleware become available. Thus, technology refresh of the implementation of the bottom half or the diagram (includes the implementation of the API) will continue whether or not the interface supplied by the ubiquitous API is modified or not. The fact that applications do not necessarily need to be modified is an advantage of the layering.

7 System Models

The issue to be discussed in this section is the nature of the interface language provided by the ubiquitous API. The realization of a ubiquitous API may seem a straightforward proposition, but the devil is in the details. The application has many very important, very complicated things to say, particularly to the compiler. The complexity of representing system modalities and configurations could very well damn the whole concept. Therefore, I will present a specific idea of how that complexity might be managed to justify a claim that the whole approach is realistic in practice as well as in theory.

The general idea is to represent a system's architecture as a state machine. A good way to imagine a state machine is as a directed graph where the nodes are the states and labeled directed edges indicate possible state transitions (Figure 4). An edge label describes when that state transition is permitted. In the model used here, the state machine is *non-deterministic*. That means, exactly, that the transition conditions of more than one edge leading away from a single node might be satisfied simultaneously. Since an actual execution of the machine will always be in a single state, this entails that some agent not in the graph must determine which transition is actually taken when there are choices.

The nodes or states in this representation are *configurations* that show software organization including process structure, branching, dataflow and control dependencies, and, in general, the sort of information that is typically expressed in an architecture description

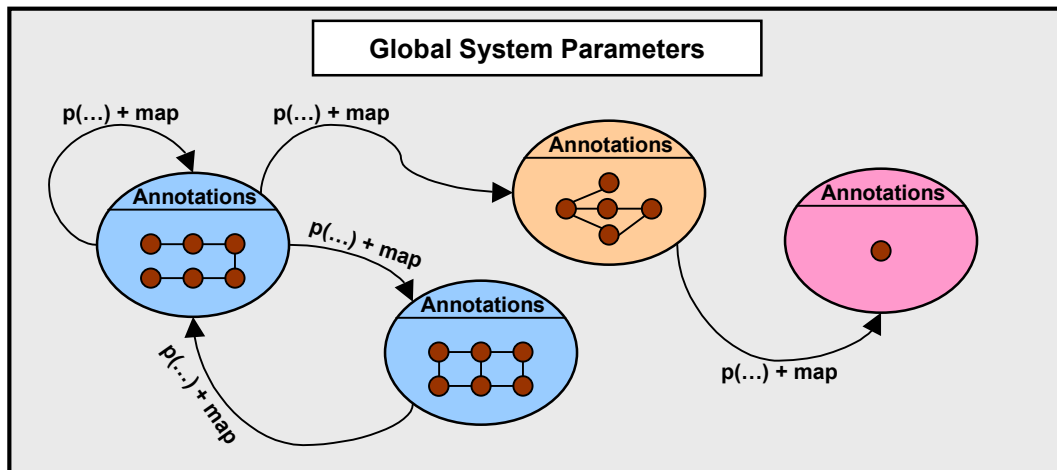


Figure 4: Application description graph: nodes are configurations; directed edges are labeled with transition predicates & maps; same-color nodes comprise a mode.

language (ADL). Nodes will also be annotated with the execution analyses gathered in the software development environment. These annotations may be parameterized, e.g., execution times as functions of CPU speed or morphed organization. The annotations

may also include assumptions such as deadline periodicity, the morphed organization, or energy availability. Note that the compiler as well as the application developers and simulation can contribute to these annotations.

The edge labels in this representation are predicates on events and a small number of global system parameters. The occurrence of an event is signaled either by the application using a specific API call provided for this purpose or by the hardware and resources layer. Examples of application signals are ATTACK MODE or TIMELINE CHANGE and an example of a resource signal is CACHE HITS BELOW THRESHOLD or PROCESS x FAILED. The ubiquitous API is responsible for checking the conditions on transitions out of the current state and choosing one of them if appropriate. If a transition is taken, the ongoing computation is reorganized using the descriptions of the FROM and TO states and the precompiled solutions provided by the compiler.³ The global system parameters such as REMAINING MISSION TIME and the like are used to determine if transitions are possible and to aid in selecting one. Dynamic transition selection and management is a primary responsibility of the ubiquitous API and the basis for its decisions and behaviors is the knowledge accumulated at compile time.

Another part of a transition specification is a map between architecture components in the FROM and TO states. Consider a morph that moves a set of processes from a mesh to a streaming configuration (e.g., the change represented by the two leftmost nodes in Figure 4). Since the same processes can appear in both worlds and must therefore retain significant portions of their states, there must be a method to specify the intended correspondence. Further, there must be a way to specify the intended stream order, i.e., which process streams its output to which other one. Without the knowledge supplied by maps, such reorganizations would not be possible. It is the responsibility of the compiler to include cost estimates and other descriptive parameters for competing alternative transitions. Cost estimates, when there are choices, will be the determining factors in most cases and the map will help make these estimates.

The logical question at this point in the description of the interface language is “What is a mode vis-à-vis the suggested representation?” The natural answer is that a mode is a collection of alternative states. Each of the states is an alternative method to accomplish a similar but not necessarily identical goal. Consider a SEEK TARGET mode as an example with two alternative states distinguished by their energy-consumption patterns. The one chosen will depend on BATTERY LEVEL and projected MISSION LIFETIME. The general goal of both is to find a suitable target. However, the specific goal of one state may be to find a high-valued target while the other would accept a secondary target if it were closer. Note that more than target-match thresholds could be different, the computational methods may be quite different in the two states also.

Since the states that comprise a mode are alternatives and will likely share portions of the predicates on the edges leading to them, a mode can be formalized by the following tactic. First determine the common condition that distinguishes a mode from other modes

³ A state can have an output edge that loops back to it. This trick makes it possible to consider the other exiting edges as *optional* transitions. Thus, a new-state transition would only occur if it were more optimal.

and define it as a *mode predicate*; second, require that exactly one mode predicate be part of the total predicate on each edge; third require that all edges leading to the same node use the same mode predicate. In other words, modes just label a partition of the states of our state machine.

This section has described the ubiquitous API and the interface language between it and the application as if all possible complications were simultaneously in play. Examples were selected to emphasize realtime features, power awareness, and morphing. The future will likely see systems that need all of this and more so that generality is a necessary feature of the approach. A prototype of a ubiquitous API could be developed that serves fewer masters and still provides reasonable feedback on the merits of the ideas. The danger is that too narrow of a focus will invite the sort of hack-and-slash results where there is no genuine possibility of generalization, even in the domain of chosen focus.

8 Related Research

The astute reader may wonder why a new language is needed rather than use an existing architecture definition language (ADL). The reason is that no existing ADL can do all that is needed here even though most provide interesting capabilities that are not altogether necessary for our purposes. However, several ADL tools, e.g., ACME [12] or CSL [3], could be adapted as our state definition language (SDL). The key to a good representation for the ubiquitous API interface is its ability to represent alternative configurations of the same mode. The emphasis of the ADL community is to represent alternative views—different abstraction—of the same configuration. The conclusion of this article is that the systems of interest will reorganize in response to dynamic events. Therefore, its designers must provide for reorganizations as a first-class application concern. The ADL world is built on a different assumption—the standard layering assumption—which we have seen won't work particularly well in the domains of interest herein.

It should be noted that the PACC and PCA communities are both keenly aware of the need for new system-building paradigms. Sponsored research includes specialty compilers, offline as well as online smart schedulers, simulation and analysis tools, and configuration managers to assist design-space exploration. PCA activities include the Morphware Forum [11] where participants are inventing architectural taxonomies and an API to intelligently support morphable systems. In addition versions of UNIX and LINUX are offered, by many sources, to address some concerns of realtime systems. Even the ADA programming language was designed to afford more application controls to those systems that needed to show compliance to rigid resource allocation requirements.

One can view the proposed ubiquitous API either as a novel approach or as an attempt to tie up loose ends. It is the latter because relevant bits and pieces of technology are available as a tech base. The novelty is its view of a strongly connected interface that extends from the software development laboratory to the execution-time environment. Information exchange between the application layers and the supporting infrastructure will occur where the information resides and offline resources that are readily available will be used to improve the efficiency and performance of the system before runtime. In this way, we will have the proper tools and engineering paradigms to develop cost-effective systems

that can adapt and reorganize themselves to provide best functionality. Thus, the resulting systems will exhibit the precious property of self-awareness so that they can adapt their behavior and methods appropriately. Self-awareness is the result of explicit representations of possible configurations, and reasons for choosing one of them over another. None of the individual tech base components can make this claim.

9 References

- [1] J. A. Barnett, "Application level power awareness," in R. Melham and R. Graybill (Eds.) *Power-Aware Computing*, Kluwer, 2002. Examples of application-specific energy optimizations at design time and runtime.
- [2] J. A. Barnett and A.S. Cooperband "Priority is a limited property," *Operating Systems Review* 17(3) 1983. Documents surprising behavior of standard semaphore implementations when priorities are supported.
- [3] J. A. Barnett, "Module Linkage and Communication in Large Systems," in D. R. Reddy (Ed.) *Speech Recognition: Invited Papers of the IEEE Symposium*, Academic Press (1975). Discusses, CSL, a language to describe system architectures through data connectivity (software pipes) and control regimes to coordinate system executions.
- [4] B. Bohem, et al, *Software Cost Estimates with COCOMO II*, Prentice Hall, 2000. Chapter 2 provides data on extraordinary costs for realtime software, systems that do considerable multiple resource scheduling, deal with device timings, or exercise distributed control.
- [5] DARPA IPTO, "Power Aware Computing and Communications Project Home Page," <http://www.darpa.mil/ipto/research/pacc/index.html>. The goals, objectives, and progress of the PACC program.
- [6] DARPA IPTO, "Polymorphous Computing Architectures Project Home Page," <http://www.darpa.mil/ipto/research/pca/index.html>. The goals, objectives, and progress of the PCA program.
- [7] DARPA IPTO, "Ubiquitous Computing Project Home Page," <http://www.darpa.mil/ipto/research/uc/index.html>. The goal, objectives, and progress of the UC program.
- [8] R. Melham and R. Graybill (Eds.), *Power-Aware Computing*, Kluwer 2002. A compendium of articles that deal with many aspects of power-aware hardware, software, middleware, and applications.
- [9] D. Parnus, "On the criteria to be used in decomposing systems into modules," *CACM* 15(12), December 1972. The article that introduced the concept of information hiding to the world.
- [10] J. Pike, "ATACMS Block II, Brilliant Anti-Armor Technology (BAT)," <http://www.globalsecurity.org/military/systems/munitions/atacms-bat.htm> 2001. A brief description of the BAT submunition and the ATACAMS supersonic dispense system.
- [11] M. Richards, Georgia Tech Research Institute, "The Morphware Forum," <http://www.morphware.org/>. The forum hosts the collaborative efforts of PCA researchers to design efficient, reusable application-to-infrastructure interfaces for morphable hardware.

- [12] School of Computer Science, Carnegie Mellon University, “The ACME architectural Description Language,” <http://www-2.cs.cmu.edu/~acme>. ACME is an unusual ADL in the sense that it is “semantic free.” It provides tools and languages to represent architectures—objects and connections—but the user provides the specific interpretations of entities.
- [13] E. F. Van de Velde, *Concurrent Scientific Computing*, Number 16 in Texts in Applied Mathematics, Springer-Verlag, 1994.

Acknowledgements

This effort is sponsored by Defense Advanced Research Projects Agency (DARPA) through the Air Force Research Laboratory, USAF, under agreement number F33615-02-C-4001. The opinions expressed are those of the author and do not necessarily reflect the opinions or conclusions of any other individual or agency.

Chapter 1

APPLICATION-LEVEL POWER AWARENESS

Jeffrey A. Barnett

Northrop Grumman Corporation

Automation Sciences Laboratory

El Segundo, California

jbarnett@nrtc.northrop.com

Abstract Optimizing resource allocation to best meet system goals is the essence of good engineering. It is the unifying principal in the design of resource-constrained systems such as aircraft, farming complexes, and even military forces. Well-designed systems accomplish tasks using minimal resources. They also dynamically adapt their methods and goals to best use available resources. This is as evident in artificial systems as it is in natural systems. Below, two examples—one a design time problem and the other a dynamic allocation problem—are developed to elucidate the sort of engineering that must be applied to resource management. The first example is a design tradeoff between the efficient use of electric power and the performance of an aircraft. The second example is dynamic allocation of a limited energy budget between cooperating sensors to maximize the quality of the combined measurement.

Keywords: Power aware design, dynamic energy management

Introduction

The most successful of nature's systems are those that have adaptations to optimally use available resources. The form factors of birds and fish, for example, are both optimized for efficient motion. Since drag, density, and buoyancy in the air and in water are different, their shapes are quite different. Thus, we observe that design optimizations are specific to environments.

Dynamic adaptation of behavior is another aspect of energy optimization. Ordinarily, animals select prey that maximize the expected return over their investment to catch it. However, in times of great need, minimal margins are readily accepted and in times of plenty, indifferent choices are made. Plants adapt their processes to such factors as the availability of sunlight, moisture,

and nutrients. Even the reproductive decisions of some species are affected by environmental cues and the state of the individual at the time. Nature has indeed developed energy-aware systems that in both design and behavior exhibit resource awareness.

Current events highlight the need for energy-aware artificial systems too. Our petroleum resources are rapidly dissipating and utility companies cannot produce enough electricity to meet all of our demands all of the time. If these were the only drivers, conservation remedies, new energy sources, and normal engineering progress would hold the problems in check.

However technology, particularly the modern computer, promises new types of systems for applications that will operate in severely energy-constrained environments. Extended duration space exploration is one well-known example. Another is the construction and deployment of unattended sensor networks comprising 1000's of miniaturized elements. Each element must be put in place with sufficient power, e.g., from a battery, to last its entire lifetime and support sensing, computing, and communications. Elements, in addition, may scavenge energy from sunlight, natural vibrations, or even chemical interactions with the environment. Energy management and optimization clearly will be of overriding importance in the design of these systems as will the need to dynamically adapt their behavior by considering the urgency of goals and the available resources.

Below, two examples of power-aware applications are described. The first is aircraft design where energy conservation contributes to overall performance by reducing battery and generator weight to enable longer flight times. Section 1 develops the mathematical relationship linking power efficiency to performance. That relationship is used to examine tradeoffs in terms of two different aircraft. Section 2 considers an application where multiple sensors develop joint measurements while sharing an energy budget. The goal is to minimize the variance of the resulting measurements while using the least energy possible and still meet system objectives. Thus, the second example is dynamic decision-making in the management of resources.

Section 3 summarizes the results and relates them to similar observations in other fields. The goal of this paper is not to present the development of specific new technology. Rather, it is to provide examples of how engineers can reason about and construct application-specific systems that will operate in energy-constrained environments.

1. A Design-Time Optimization for Aircraft

The value of power awareness and energy reduction technologies are determined by the specifics of an application domain and the milieu in which its systems will operate. Intelligent power management is one of the key technology

enablers to successful deployment of deep space probes, networks comprised of miniature battery-powered sensors, and the Land Warrior. Below, a simple relation between energy efficiency and maximum flight time (endurance) for aircraft is derived. That relation is used to model the value of efficient design as measured by increased endurance: Sometimes it is very valuable and sometimes it is not. The point of this exercise is not to solve aircraft problems per se. Rather, it is to introduce the sort of domain-specific analysis that is necessary to make design-time tradeoffs for an application.

1.1 Domain Considerations

Weight is the enemy of air vehicle performance. Heaviness directly decreases maneuverability, handling, range, and endurance, where endurance is maximum flight time. Since savings in the use of electrical energy will reduce the weight of the generator, the battery, or both, power awareness will increase the missions that a given aircraft can undertake and complete. Another benefit of weight reduction is that the weight budget for the fuel used to power the air vehicle can be increased when the battery or generator weight is decreased.

Reducing electric power consumption also decreases heat production. Therefore, more equipment can be packed into the same or smaller form factor. Since decreasing size increases maneuverability, range, and endurance, we have an additional argument for power awareness. Further, obvious benefits accrue when power-aware avionics can compensate for unusual situations such as generator loss. This example makes the point that power awareness is more than energy minimization: it's about doing the best you can with the resources that are actually available.

Concerns about the effects of electric energy consumption on aircraft performance are not new. Davis (1946), for example, discusses space, and heat considerations along with their impacts on operational costs. Power awareness is even more important in the modern world. One example is the Helios, a product of AeroVironment Corporation, with the goal of indefinite sustained flight powered by scavenging solar energy. Other extreme examples are smart munitions that either have no propulsion system or only use propulsion in burst modes. Since there is no engine-powered generator, all electric energy must be supplied by batteries to support sensors, computation, communications, and control surface actuators. Batteries are a limiting resource that produce unwanted heat, consume precious space, and add weight.

The standard relation approximating the endurance achievable by an aircraft as a function of its weight and the amount of fuel it carries is derived next. The following discussion extends that relation to include the effects of electric power utilization efficiency, hence battery weight, on endurance. Finally, that

relation is used to analyze the effects of power efficiency on endurance for two example aircraft.

1.2 Endurance is a Function of Weight and Fuel

The relation between endurance and an air vehicle's weight and the amount of fuel it carries is derived here. A reasonable first-order approximation is used to simplify the mathematics: instantaneous fuel consumption rate is linearly related to total vehicle weight for a constant velocity and altitude. The next section extends the relation to consider the effects of electric energy conservation on weight, hence, its relation to endurance. See Raymer (1992) for justification of the assumptions made here as well as derivations of more detailed relationships when other variables are considered.

Let $f = f(t)$ be the amount of fuel remaining at flight time t and let $f_0 = f(0)$ be the amount of fuel loaded on an aircraft that weights W_d dry (fuel weight is zero). The assumption is that the amount of fuel, measured by weight, necessary per second of flight time per unit of aircraft weight is a constant. Thus,

$$f' = -c(W_d + f)$$

where $f' = df/dt$ and $c > 0$ is the fuel consumption constant. The solution to this simple ode, subject to the constraint that $f(0) = f_0$, is

$$f(t) = W_0 e^{-ct} - W_d,$$

where $W_0 = W_d + f_0$ is the initial weight of the aircraft at $t = 0$.

The endurance, E , is defined to be the maximum flight time. Clearly, $f(E) = 0$ is the condition to find that unique value of E . Therefore,

$$E = \frac{1}{c} \times \log \left(\frac{W_0}{W_d} \right). \quad (1.1)$$

An explanation, in part, for this, perhaps unexpected result, is that in order to increase endurance, more fuel is obviously needed. But, additional fuel is necessary to carry the original extra fuel, etc. Thus, weight has a nonlinear effect on endurance. Note, this derivation is a reasonable approximation for many land as well as airborne vehicles.

1.3 Endurance is a Function of Energy Conservation

In order to extend the above derivation to account for a battery and, hence, to account for the efficiency with which that battery is used, a few details must be added to our simple model. Let W_S be the structural weight of the aircraft and W_B be the weight of its battery. Thus,

$$\begin{aligned} W_d &= W_S + W_B \\ W_0 &= W_S + W_B + f_0. \end{aligned}$$

In order to focus on the relevant tradeoff for this discussion—battery weight versus fuel—define $L = W_B + f_0$ as the total weight budget for fuel plus battery and β as the battery weight needed for one unit of mission time. Thus, $W_B = \beta E$. β is interpreted as the efficiency of the *energy consumers* not of the *battery*, though the analyses are practically identical in either case. Now,

$$\begin{aligned} W_d &= W_S + \beta E \\ W_0 &= W_S + L, \end{aligned}$$

where the first equation assumes that there is just enough battery to last for the whole mission. In other words, L is split between fuel and battery weight so that both resources expire simultaneously. There is one important difference between fuel and battery that makes these equations non-symmetric: when fuel is consumed, its weight decreases but no such reduction will be observed for batteries. Substituting the formulas into (1.1) yields

$$E = \frac{1}{c} \times \log \left(\frac{W_0}{W_S + \beta E} \right).$$

What we desire is an expression for E as a function of β so that the payoff of efficient energy utilization can be measured in terms of endurance enhancement. A quick inspection shows that an elementary expression isn't available. However, it is straightforward to express β as a function of E :

$$\beta = \frac{W_0 - W_S \exp(cE)}{E \exp(cE)}. \quad (1.2)$$

Clearly, the value of E must be positive and is bounded from above by the case where $f_0 = L$ and, hence, $\beta = 0$. Thus,

$$0 < E < \frac{\log(W_0) - \log(W_S)}{c}. \quad (1.3)$$

Equation 1.2 and the bounds on E are used below to numerically generate and graph (β, E) pairs for some examples.

1.4 Aircraft Examples & Analysis

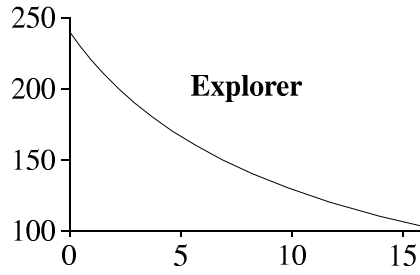
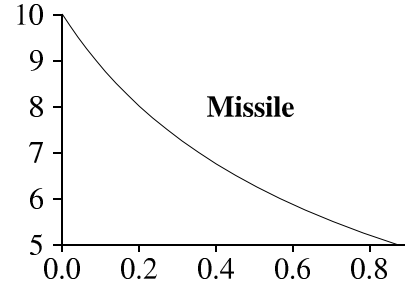
Two simple examples of vehicles designed for quite different mission profiles are analyzed in this section. The first, called Explorer, is a simple unmanned sensor craft that uses a generator to support most applications. It flies missions of a few hours duration and its battery is primarily for power during emergencies, e.g., for flight recorders or engine restarts after flame outs. The second example, called Missile, is a smart munition that uses batteries to power all onboard electronics. Fuel is only used in burst mode to regain altitude so as to increase

Table 1.1. Parameters defining two aircraft examples.

Example	c	W_s	L	W_0	E_{\max}
Explorer	.00155/min	6,900 lb	3,100 lb	10,000 lb	240 min
Missile	.05108/min	15 lb	10 lb	25 lb	10 min

target seek time. Its mission times are of the order of a few minutes. Table 1.1 summarizes the relevant parameters for these two hypothetical examples and $E_{\max} = \log(W_0/W_s)/c$ is the maximum endurance from (1.3).

E versus β pairs have been generated using (1.2) and are displayed in Figures 1.1a and 1.1b. Endurance, E , clearly increases as electric power utilization, as measured by β , decreases. There is no way to tell, at this point in the

Figure 1.1a. E versus β relation for the Explorer example.Figure 1.1b. E versus β relation for the Missile example.

development of the two examples, whether power-saving measures would be worthwhile or not. We need to know the current β , the merit of a ΔE measured as an increased fraction of successful missions, and the cost of developing and inserting power-aware technology that can provide the ΔE in order to make that determination.

Such details are beyond the scope of this paper though a simple heuristic analysis is presented. Figures 1.2a and 1.2b suggest the shapes of typical

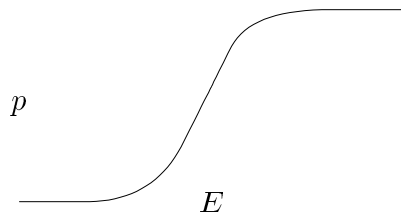


Figure 1.2a. Probability of mission success as a function of endurance.

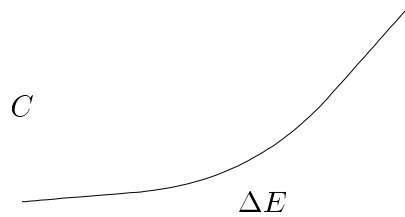


Figure 1.2b. Cost to increase endurance as a function of the desired increment.

tradeoff curves: $p = p(E)$ is the probability of a successful mission if endurance is E and $C = C(\Delta E, E)$ is the cost to achieve an incremental endurance enhancement given the current value of E . Design tradeoffs are then formed in terms of this and other relevant information.

The Explorer example posits that batteries are not the main source of electric power and it is fair to assume that its designers made various decisions to maximize endurance. In other words, it is unlikely that the total weight of the generator and battery are a significant fraction of the total vehicle weight. Further, $f_0 \approx L$. Therefore, power-awareness, even if it could reduce the generator weight cannot increase endurance by more than a few minutes.

The Missile example is completely different. Battery weight is a significant fraction of L and L is a significant fraction of W_0 . Once again, conservation can only increase endurance by a few minutes but a few minutes, compared to a current endurance of less than 10 minutes, could make all the difference in whether or not missions are successful.

1.5 Beyond Design-Time Optimization

The results and examples developed above are meant to show how considerations of power utilization can impact overall system performance. In particular, the effects of energy conservation on maximum flight time are demonstrated. Energy conservation, however, is only one aspect of power awareness. The more general problem is to dynamically determine how to optimize mission goal achievement as a function of the resources actually available. This is a much more difficult problem and is even more of an application-dependent issue than is straightforward energy minimization.

The Missile example suggest a plethora of interesting power management problems. For example, given a projected time to target impact maximize the probability of a successful strike. The problem is how best to use the remaining battery energy during the known flight time. Should sensor resource allocations be increased or should communications and computation be used to receive and fuse an ally's sensor data with onboard information? The answer depends on the resources available as well as the relative quality and costs of the two solutions. In other words, real power awareness depends on the ability to make dynamic, situation-dependent as well as application-dependent decisions. The sort of design-time tradeoffs analyzed herein are important too, but they are only a first step toward realization of fully power-aware systems.

2. Dynamic Energy Allocation for Cooperating Sensors

The problem considered in this section is how to split a limited energy budget between several sensors whose measurements will be fused. The objective is to minimize the variance of that joint measurement. A simple fusion model is

defined, measurement variance is related to energy expenditures, and a criterion for an optimal allocation is derived. An extended example of a dynamic allocation problem and its solution is presented in three parts: (1) a specific model of sensor variance as a function of energy allocation is introduced, (2) the optimality problem for two sensors described by that particular model is solved, and (3) a numerical example is used to illustrate an optimal policy and the sort of decision-making that it engenders. This note is meant as an example of how one might formulate and solve dynamic energy management problems and not as a definitive practical result.

2.1 Fusing Sensor Measurements

If sensors $1, \dots, n$ make measurements m_1, \dots, m_n that are statistically independent, the joint estimate m is taken to be

$$m = \frac{w_1 m_1 + \dots + w_n m_n}{w_1 + \dots + w_n}, \quad (1.4)$$

where $w_i = v_1 \dots v_n / v_i$ and v_i is the variance of measurement m_i . The variance, v , of the joint estimate is

$$v = \frac{1}{\frac{1}{v_1} + \dots + \frac{1}{v_n}}. \quad (1.5)$$

The function, $v = v(v_1, \dots, v_n)$, has the following intuitive properties:

Less is better: $\frac{\partial v}{\partial v_i} \geq 0$.

Progress: $v(v_1, \dots, v_n) \leq v_i$.

As good as it gets: $v(v_1, \dots, v_n) = 0$ if any $v_i = 0$, a corollary of progress.

No pain, no gain: $v(v_1, \dots, v_n) = v(v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_n)$ if $v_i = \infty$.

Unit independence: $v(rv_1, \dots, rv_n) = rv(v_1, \dots, v_n) = 0$.

du Plessis (1967) provides some justification for the formulas assumed here as well as an introduction to the general topic of fusion using Kalman filters. Note, fusion formulas such as (1.4) and (1.5) are more complicated if the sensor measurements are statistically dependent or if multiple features are measured as is often the case.

2.2 Energy Allocation Determines Sensor Performance

Modern sensors offer many controls that trade energy for performance. Examples are raw power, pulse width, number of pulses per measurement, frequency of measurement, and even spectrum choices for hyperspectral sensors.

Some passive sensors can vary the number of pixels or the parts of the spectrum that are processed. The more energy used, the higher the quality of the measurement. Below, quality is grossly summarized by the resulting variance of the measurement. Specific issues such as precision, etc., are assumed to be sufficiently captured by the variance estimate.

Let $v_s(e)$ represent the variance expected when sensor s makes a measurement using energy $e \geq 0$. Such a function should satisfy these criteria:

Positivity: $v_s(e) > 0$.

More is better: $\frac{dv_s}{de} \leq 0$.

No free lunch: $v_s(0) = \infty$.

The astute reader may be troubled by the *no free lunch* property. Isn't there always some *a priori* information available so that the initial variance would be finite? For example, an angle measurement is always in the interval $[0, 2\pi)$. The way to analyze the situation is to assume that the variance of the *a priori* estimate is v_0 and to combine it with v_1, \dots, v_n to get the combined variance, $v = (v_0^{-1} + v_1^{-1} + \dots + v_n^{-1})^{-1}$. Since the optimality criterion derived in the next section is the same whether v_0 is accounted for or not, the math is simplified by ignoring the presence of *a priori* estimates.

2.3 Minimizing Variance Through Energy Allocation

Given a total energy budget, e , to be used to obtain independent measurements m_1, \dots, m_n , the obvious objective is to divide it so that the resulting variance is minimized. Assume that $e_i \geq 0$ is used for the measurement made by sensor i . The problem is to minimize (1.5) subject to the constraint that $e_1 + \dots + e_n = e$. Simply use Lagrangian multipliers as follows:

$$v(e_1, \dots, e_n, \lambda) = \frac{1}{\frac{1}{v_1} + \dots + \frac{1}{v_n}} + \lambda \left(e - \sum e_i \right).$$

Necessary minimization criteria follow from $dv/de_i = 0$:

$$\begin{aligned} 0 &= \frac{dv}{de_i} \\ &= \frac{-v'_i/v_i^2}{\left(\frac{1}{v_1} + \dots + \frac{1}{v_n} \right)^2} - \lambda \\ \frac{v'_i}{v_i^2} &= -\lambda \left(\frac{1}{v_1} + \dots + \frac{1}{v_n} \right)^2, \end{aligned}$$

where $v'_i = dv_i/de_i$. Since λ does not depend on i , the entire right hand side is independent of i . Therefore,

$$\frac{v'_i}{v_i^2} = \frac{v'_j}{v_j^2}, \quad (1.6)$$

for all $1 \leq i, j \leq n$. Standard dynamic programming techniques can be used to develop optimal allocations when there are more than two sensors.

2.4 Applying the Theory

This section develops an example of variance minimization through energy allocation. The problem of achieving a given result quality with minimum resource investment is addressed in the following section. Both problems are examples of dynamic energy management. The discussion proceeds in three parts: (1) a specific, parameterized sensor model that defines variance as a function of energy is introduced, (2) the two-sensor minimization problem is solved, and (3) the optimal allocation policy and the variance achieved are analyzed for particular sensor instances.

2.4.1 A Parameterized Sensor Model. The form of v_i assumed in the optimization example developed in the rest of this section is

$$v_i(e) = a_i + b_i/e, \quad (1.7)$$

where $a_i, b_i > 0$. Such forms obviously possess the properties required above for variance functions. For future reference, note that

$$\frac{dv_i}{de} = \frac{-b_i}{e^2}. \quad (1.8)$$

The parameters a_i and b_i have the following interpretations in this formulation: a_i is the limit of the sensor's operation, i.e., $a_i = v_i(\infty)$, and b_i is the difference between the sensor's variance using one unit of energy and its best possible performance, i.e., $b_i = v_i(1) - v_i(\infty)$.

2.4.2 Solving the Two-Sensor Problem. Assume two independent sensors whose behavior is defined by (1.7) and a total energy budget of e . Let energy $0 \leq x_1 \leq e$ be allocated to the first sensor and $x_2 = e - x_1$ be allocated to the second. Then from (1.7) and (1.8),

$$\begin{aligned} v_1(x_1) &= a_1 + b_1/x_1 \\ v_2(x_2) &= a_2 + b_2/(e - x_1) \\ v'_1(x_1) &= -b_1/x_1^2 \\ v'_2(x_2) &= -b_2/(e - x_1)^2. \end{aligned}$$

Now use criterion (1.6) to find the optimal allocation as follows:

$$\begin{aligned}
 \frac{v'_1}{v_1^2} &= \frac{v'_2}{v_2^2} \\
 \frac{-b_1/x_1^2}{(a_1 + b_1/x_1)^2} &= \frac{-b_2/(e - x_1)^2}{(a_2 + b_2/(e - x_1))^2} \\
 \frac{b_1}{(a_1 x_1 + b_1)^2} &= \frac{b_2}{(a_2(e - x_1) + b_2)^2} \\
 \sqrt{b_1}(a_2(e - x_1) + b_2) &= \sqrt{b_2}(a_1 x_1 + b_1).
 \end{aligned}$$

The solutions to this linear equation for the optimal x_1^* and $x_2^* = e - x_1^*$ are

$$x_1^*(e) = \frac{a_2\sqrt{b_1}e + b_2\sqrt{b_1} - b_1\sqrt{b_2}}{a_1\sqrt{b_2} + a_2\sqrt{b_1}} \quad (1.9a)$$

$$x_2^*(e) = \frac{a_1\sqrt{b_2}e + b_1\sqrt{b_2} - b_2\sqrt{b_1}}{a_1\sqrt{b_2} + a_2\sqrt{b_1}}. \quad (1.9b)$$

Since x_1^* and x_2^* must be nonnegative, the numerators must be nonnegative. Thus, there are two validity constraints entailed by (1.9a) and (1.9b):

$$\begin{aligned}
 e &> \frac{\sqrt{b_1 b_2} - b_2}{a_2} \\
 e &> \frac{\sqrt{b_1 b_2} - b_1}{a_1}
 \end{aligned}$$

If the first constraint is violated, the optimal allocation is $x_1^* = 0$ and $x_2^* = e$. If the second is violated, the optimal allocation is $x_1^* = e$ and $x_2^* = 0$. In the first case the resultant minimum fused variance, v^* , is $v^*(e) = a_2 + b_2/e$ and in the second the minimum variance is $v^*(e) = a_1 + b_1/e$. When there is an “interior” solution, v_1^* and v_2^* are calculated by substituting (1.9a) and (1.9b) into (1.7) to get

$$\begin{aligned}
 v_1^*(x_1) &= \frac{a_1 a_2 e + a_1 b_2 + a_2 b_1}{a_2 e + b_2 - \sqrt{b_1 b_2}} \\
 v_2^*(x_2) &= \frac{a_1 a_2 e + a_1 b_2 + a_2 b_1}{a_1 e + b_1 - \sqrt{b_1 b_2}}.
 \end{aligned}$$

These values are substituted into (1.5) and simplified to calculate the resultant minimum total variance as

$$v^*(e) = \frac{a_1 a_2 e + a_1 b_2 + a_2 b_1}{(a_1 + a_2)e + (\sqrt{b_1} - \sqrt{b_2})^2}.$$

Table 1.2. Formulas for optimal energy allocations and resulting variances.

	$0 \leq e < \frac{\sqrt{b_1 b_2} - b_2}{a_2}$	$0 \leq e < \frac{\sqrt{b_1 b_2} - b_1}{a_1}$	Otherwise
$x_1^*(e) =$	0	e	$\frac{a_2 \sqrt{b_1} e + b_2 \sqrt{b_1} - b_1 \sqrt{b_2}}{a_1 \sqrt{b_2} + a_2 \sqrt{b_1}}$
$v_1^*(e) =$	∞	$a_1 + b_1/e$	$\frac{a_1 a_2 e + a_1 b_2 + a_2 b_1}{a_2 e + b_2 - \sqrt{b_1 b_2}}$
$x_2^*(e) =$	e	0	$\frac{a_1 \sqrt{b_2} e + b_1 \sqrt{b_2} - b_2 \sqrt{b_1}}{a_1 \sqrt{b_2} + a_2 \sqrt{b_1}}$
$v_2^*(e) =$	$a_2 + b_2/e$	∞	$\frac{a_1 a_2 e + a_1 b_2 + a_2 b_1}{a_1 e + b_1 - \sqrt{b_1 b_2}}$
$v^*(e) =$	$v_2^*(e)$	$v_1^*(e)$	$\frac{a_1 a_2 e + a_1 b_2 + a_2 b_1}{(a_1 + a_2)e + (\sqrt{b_1} - \sqrt{b_2})^2}$

Table 1.2 summarizes these results for the optimal allocation policy: $x_1^*(e)$ is the optimal allocation of total energy, e , to the first sensor and $v_1^*(e) = v_1(x_1^*(e))$ is the variance it achieves. The functions x_2^* and v_2^* are similarly defined for the second sensor. Finally, $v^*(e) = v(v_1^*(e), v_2^*(e))$ is the minimized variance of the fused measures given total energy budget e .

2.4.3 Applying the Results. This section develops a simple numerical example with two sensors. The resulting allocation policy and variance as a function of energy curves exhibit some interesting behaviors. Formulas for the optimal allocation policy and the resulting variances, derived from Table 1.2 using the constants $a_1 = 4$, $b_1 = 8$, $a_2 = 3$, and $b_2 = 18$, are shown in Figure 1.3a.

The optimal allocation of energy to the sensors is depicted in Figure 1.3b. Initially when $0 \leq e \leq 1 = (\sqrt{b_1 b_2} - b_1)/a_1$, all of the available energy is allocated to the first sensor; when $e > 1$, it is split between the two sensors. The first sensor continues to receive a larger allocation, i.e., $x_1^*(e) > x_2^*(e)$, while $0 < e < 4$ but there is a role reversal when $e > 4$ where $x_1^*(e) < x_2^*(e)$. When $e = 4$, $x_1^*(e) = x_2^*(e) = 2$.

The variances achieved using the optimal allocation policy are shown in Figure 1.3c where a log-valued vertical distance scale has been employed to improve visualization. The first sensor gives better results (has a lower variance) when a relatively small amount of energy is available but its limiting behavior is not as good as the second sensor. The variance of the fused measure, v^* , coincides with v_1^* when $0 \leq e \leq 1$. From then on it is lower. The contribution from the first sensor has lower variance than the second, i.e., $v_1^*(e) < v_2^*(e)$,

$$\begin{aligned}
 x_1^*(e) &= \begin{cases} e & e \leq 1 \\ \frac{e+2}{3} & e > 1 \end{cases} & v_1^*(e) &= \begin{cases} 4 + 8/e & e \leq 1 \\ \frac{4e+32}{e+2} & e > 1 \end{cases} \\
 x_2^*(e) &= \begin{cases} 0 & e \leq 1 \\ \frac{2e-2}{3} & e > 1 \end{cases} & v_2^*(e) &= \begin{cases} \infty & e \leq 1 \\ \frac{3e+24}{e-1} & e > 1 \end{cases} \\
 v^*(e) &= \begin{cases} 4 + 8/e & e \leq 1 \\ \frac{12e+96}{7e+2} & e > 1 \end{cases}
 \end{aligned}$$

Figure 1.3a. Optimal allocation and resulting variances for the example where $a_1 = 4$, $b_1 = 8$, $a_2 = 3$, and $b_2 = 18$.

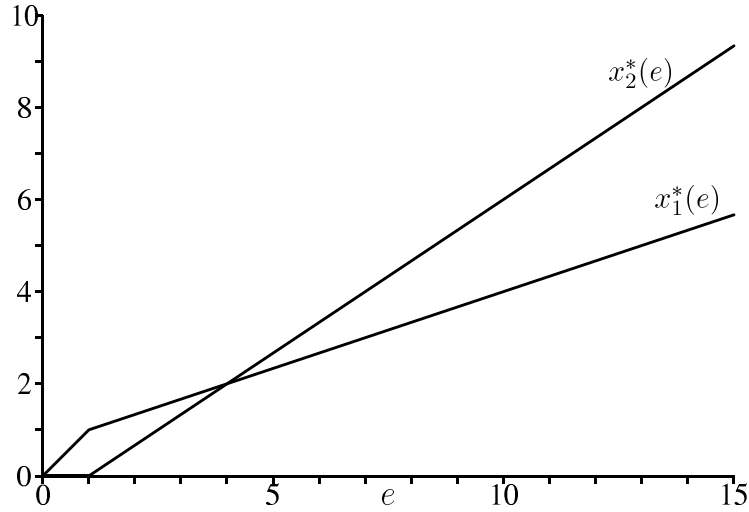


Figure 1.3b. Optimal resource allocation for the example.

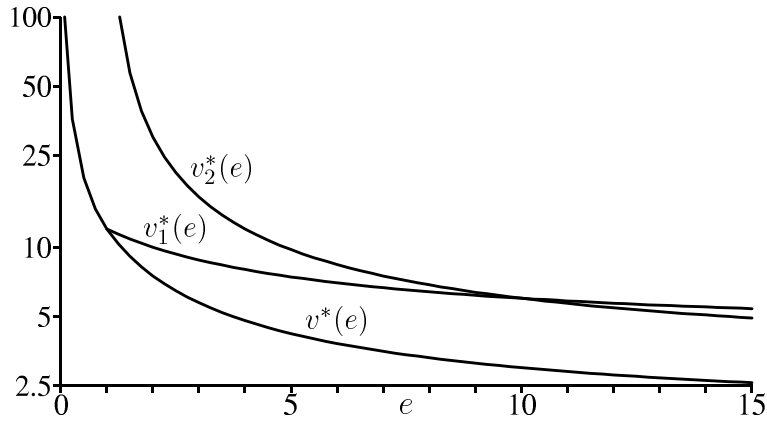


Figure 1.3c. Optimal variances for the example.

while $0 < e < 10$. However, $v_1^*(10) = v_2^*(10) = 6$ and the second sensor's contribution finally dominates when $e > 10$.

This example was deliberately contrived to exhibit crossovers in both the optimal resource allocation policy and the quality of the results as shown in Figures 1.3b and 1.3c. The purpose of this ploy was two fold: (1) to indicate that even simple models can lead to complex or unexpected behavior and (2) to make the point that straightforward analytic methods can usually derive or approximate the actual optimal policy to be applied dynamically.

2.5 Related Problems

The area of dynamic sensor management and scheduling comprises many related problems. For example, the problem may be to develop an estimate that is good enough, i.e., to use the least energy to achieve a specific variance. This problem can be solved using formulas such as those in Table 1.2. The necessary minimum energy, e^* , is found from the inverse of v^* . This is straightforward because v^* is a necessarily monotonic function. The appropriate allocations are then calculated as the values of $x_1^*(e^*)$ and $x_2^*(e^*)$.

In a more realistic scenario, the analyst would need to account for the fact that measurements are not statistically independent, as assumed above, consider additional tradeoffs, and deal with measurement sources whose analytic descriptions are different than one another, e.g., it might be more economical to use communications to obtain an ally's measurement and fuse it than to use some or all of the local sensors.

Hong and Wong (1998) and Kirubarajan, et al, (1998) describe problems related to those discussed above. Since some sensors can be time multiplexed amongst several applications, more applications can be serviced if the fraction of the sensor resources needed for each is reduced. They use a combination of filtering (fusion) and modelling. The models, formed from filter output parameters, predict target behavior between sensor measurements. The models also predict the variance increase as time of model use increases. That information is used dynamically to minimize the product of measurement duration and update rate for each application while maintaining track quality. Such techniques could be adapted by power-aware sensor schedulers. Blair, et al, (1998) describe a benchmark facility to compare algorithms of this sort.

Dynamic reasoning about the use of scarce resources is becoming more important as time goes by. It is an area that will pose many interesting science and engineering challenges in the future.

3. Afterword

Much of our understanding of the world around us is gained through hypotheses about how systems manage energy. Physicists assume least energy

principals to form laws about the universe in the large and interactions of sub-atomic particles in the small. Biologists explain life, in part, by showing how organisms scavenge energy from the environment and store it for their own use at later times.

So concerns about power-aware systems are certainly not new. What is different today is that engineers are proposing and building systems whose functioning is all about energy management. Deep-space exploration and large unattended miniature sensor networks are two examples mentioned above. In these and other systems of the same ilk, power awareness is a primary design and operational principal, not just another support technology.

It may surprise the reader to know that Sigmund Freud proposed an energy minimization principal, as part of a cognitive economy model, to partially explain the workings of the human mind. He described dreaming and humor processes through a minimization of a quantity called *psychical energy*. I will close with a quote from Freud (1905) that seems particularly germane to our current topic:

"I may perhaps venture on a comparison between psychical economy and a business enterprise. So long as the turnover in the business is very small, the important thing is that outlay in general shall be kept low and administrative costs restricted to the minimum. Economy is concerned with the absolute height of expenditure. Later, when the business has expanded, the importance of the administrative cost diminishes; the height reached by the amount of expenditure is no longer significant provided that the turnover and profits can be sufficiently increased. It would be niggling, and indeed positively detrimental, to be conservative over expenditures on the administration of business. Nevertheless, it would be wrong to assume that when expenditure was absolutely great there would be no room left for the tendency to economy."

Acknowledgments

This effort is sponsored by Defense Advanced Research Projects Agency (DARPA) through the Air Force Research Laboratory, USAF, under agreement number F30602-00-1-0511. The opinions expressed are those of the author and do not necessarily reflect the opinions or conclusions of any other individual or agency.

References

- W. D. Blair, G. A. Watson, T. Kirubarajan, and Y. Bar-Shalom, Benchmark for radar allocation and tracking in ECM, *IEEE Transactions on Aerospace and Electronic Systems*, 34 (4) (1998).
- W. W. Davis, *Cargo Aircraft*, Pitman Publishing, 1946.

- S. Freud, *Jokes and their Relation to the Unconscious*, Leipzig and Vienna: Deuticke 1905. J. Strachey, translator, Norton Library, 1963.
- T. Kirubarajan, Y. Bar-Shalom, W. D. Blair, and G. A. Watson, Management and tracking benchmark with ECM, *IEEE Transactions on Aerospace and Electronic Systems*, 34 (4) (1998).
- S. M. Hong and Y. H. Jung, Optimal scheduling of track updates in phased array radars, *IEEE Transactions on Aerospace and Electronic Systems*, 34 (3) (1998).
- R. M. du Plessis, *Poor Man's Explanation of Kalman Filtering or How I Stopped Worrying and Learned to Love Matrix Inversion*, Rockwell International, 1967. (Reprinted by Taygeta Scientific Incorporated, 1996; can be purchased at <http://www.taygeta.com>.)
- D. P. Raymer, *Aircraft Design: A Conceptual Approach*, AIAA Educational Series, (Ed.) J. S. Przemieniecki, AIAA Press, 1992.

Distributable Computations: A Conjecture

Jeffrey A. Barnett*

Northrop Grumman Corporation

Contact Information

Address: 5215 Sepulveda 1A
Culver City, CA 90230

E-mail: jbb@notatt.com

Phone: (310) 390-0353 (eves)
(310) 332-8930 (days)

*This effort is sponsored by Defense Advanced Research Projects Agency (DARPA) through the Air Force Research Laboratory, USAF, under agreement number F33615-02-C-4001.

Abstract

Distributing computations is desirable both to shorten the time necessary to get an answer and to reduce the bandwidth necessary to communicate distributed data. This note is about the latter motivation. A model of bandwidth-reducing distributed computation is developed and a conjecture about the necessary and sufficient conditions that a function evaluation can be so distributed is formed. The condition is used to construct a simple distributability test for the class of monotonic functions and that test is shown to be valid. The status of the conjecture for other classes of functions is unknown.

Keywords: Distributed computing models, bandwidth reduction computation, distributed functions.

1 Summary

Distributed computing architectures are often developed to reduce the time necessary to perform calculations. In recent times for example, computer owners world wide have donated networked resources to the search for large Mersenne primes. That effort has substantially increased the rate of finding these primes. Other examples are large numeric codes applied to complex problems ranging from weather prediction to structural analysis, biological simulation, and computational fluid dynamics.

Another motivation for distributing a computation is that the input data is distributed. Often, a tradeoff exists between local data-reduction computations and the use of additional bandwidth to ship the raw data. An unattended ground sensor network limited by battery and scavenged energy resources is an example. Is it more energy efficient to process sensor data locally and transmit only features to a central fusion site or should the sensor data be transmitted? Other examples include relatively straightforward statistical calculations on large distributed data sets or accumulating votes where communications are the major costs.

This note investigates the question of which function evaluations provide local-computation versus communication-bandwidth tradeoffs. Not all do as will be shown by examples. A conjecture about necessary and sufficient conditions that the computation of a function can be distributed to reduce bandwidth is stated below. That conjecture concerns equivalence classes induced by a partition of a function's domain. The elements, x_1 and x_2 , are in the same class if $f(x_1, y) = f(x_2, y)$ for all y , where x_1 , x_2 , and y are tuples of data. The conjecture is formed in terms of the “dimensionality” of

a surface that intersects each equivalence class induced by the partition.

The next section presents a simple example, the computation of the mean and variance of a sample split between two sites, to motivate the formal definition of distributable computation introduced in Section 3. Section 4 makes a conjecture, that if true, would provide an alternative characterization of distributable functions. Section 5 then casts the conjecture in terms of the domain partition and a set of representatives for elements of the partition. In addition, it is noted why proving the conjecture might be difficult. Finally, Section 6 examines the class of monotonic functions where the conjecture is true and, therefore, a straightforward method to check whether a computation is distributable or not is available.

2 An Example

An example is used to motivate the definition of distributed computation introduced below. The computation to be distributed is (A, V) , where A is the average and V is the variance of the numbers s_1, \dots, s_n :

$$A = \sum_{i=1}^n s_i/n \quad V = \sum_{i=1}^n (s_i - A)^2/n.$$

These results can easily be computed from the two quantities α and β , where

$$\alpha = \sum_{i=1}^n s_i \quad \beta = \sum_{i=1}^n s_i^2,$$

by the formulas $A = \alpha/n$ and $V = \beta/n - A^2$. Assume that the numbers s_1, \dots, s_k are resident at one site while s_{k+1}, \dots, s_n are resident at another. One possibility is to transmit the k -tuple, (s_1, \dots, s_k) , to the second site where A and V will be computed. However, there is a more economical

possibility shown in Figure 1. Calculate α_k and β_k at the first site (CPU₁)

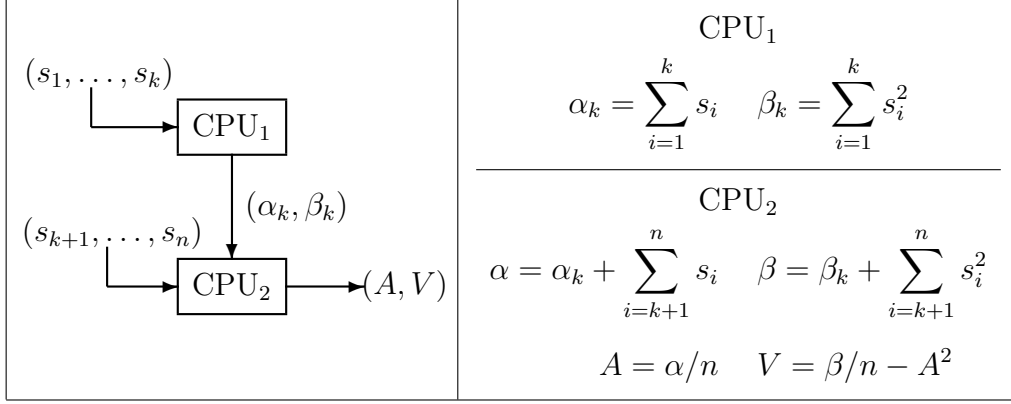


Figure 1: Distributed computation of average and variance of s_1, \dots, s_n .

and transmit only these two quantities to the second site (CPU₂) where α and β , then A and V are computed. In this example, the bandwidth is reduced by a factor of $1 - 2/k$ which can be considerable if k is large. Achieving this sort of reduction is the motivation for the distributed computation model described next.

3 Model of Distributed Computation

The following definition, as depicted in Figure 2, is meant to capture the idea of distributing a computation to reduce bandwidth.

Definition 1 *$f: X \times Y \rightarrow T$ is Z -distributable in X if there is a continuous onto $d: X \rightarrow Z$ and a continuous $c: Z \times Y \rightarrow T$, such that $c(d(x), y) = f(x, y)$ for all $x \in X$ and $y \in Y$.*

The distributed portion of the computation is d and the central site where the evaluation of f is completed is represented by c .

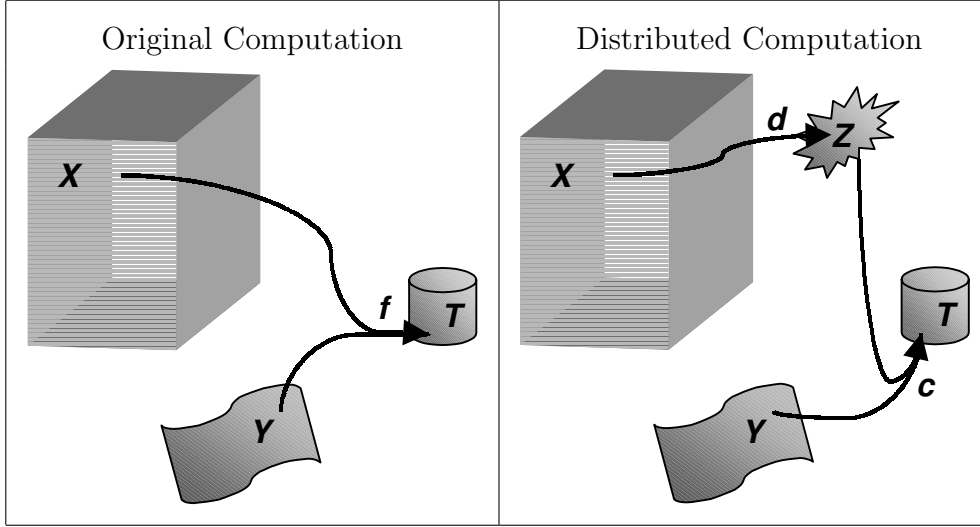


Figure 2: Computation $f(x, y)$ is Z -distributed as $c(d(x), y)$.

The definition entails that f be continuous because it is a composition of continuous functions and that Z be connected when X is connected because d is continuous. (Connectivity of X is assumed below.) Continuity of functions is required for two reasons. The first is that *computation* of discontinuous functions is rare and somewhat ill-defined because computation entails truncation and, hence, approximations. The second is that *communications* of data by a discontinuous function—say by a reduction operation from \mathbb{R}^n to \mathbb{R} that interlaces decimal digits—is not germane to actual computations and finite bandwidth resources.

The cases of interest are those where $X \subset \mathbb{R}^n$, $Z \subset \mathbb{R}^u$, and $u < n$. Of course there is always a Z such that f is Z -distributable if $u = n$. The integer u is taken to measure the bandwidth and, hence, the efficiency of the distribution strategy using c and d to distribute the computation of f . A sharper definition is suggested though it isn't used below.

Definition 2 *The function $f: X \times Y \rightarrow T$ is exactly Z -distributable in X ,*

where $Z \subset \mathbb{R}^u$, if it is Z -distributable and there is no $v < u$ and $Z' \subset \mathbb{R}^v$ such that it is Z' -distributable.

Consider the example from the previous section in terms of the first definition: Let $X = \mathbb{R}^k$, $Y = \mathbb{R}^{n-k}$, $T = \mathbb{R} \times \mathbb{R}^+$, where \mathbb{R}^+ is the nonnegative reals, and $Z = \mathbb{R} \times \mathbb{R}^+$; then $d(s_1, \dots, s_k) \rightarrow (\alpha_k, \beta_k)$ and $c((\alpha_k, \beta_k), s_{k+1}, \dots, s_n)$ are the computations shown in Figure 1.

The median is an example of a function that is difficult to distribute to advantage. It is defined as $\text{med}(x_1, \dots, x_{2n+1}) = x_{j_{n+1}}$, where j_1, \dots, j_{2n+1} is a permutation of $1, \dots, 2n+1$, such that $x_{j_i} \leq x_{j_{i+1}}$. Consider a potential distributed computation,

$$\text{med}(x_1, \dots, x_{2n+1}) = c(d(x_1, \dots, x_{n+1}), x_{n+2}, \dots, x_{2n+1}),$$

where $d: \mathbb{R}^{n+1} \rightarrow \mathbb{R}^v$. The question is, how small can v be? The claim is that $v \geq n+1$ because x_{n+2}, \dots, x_{2n+1} can be chosen such that *any one* of x_1, \dots, x_{n+1} is the median. Since no continuous d can exactly encode independent x_1, \dots, x_{n+1} in less than $n+1$ real numbers, the claim follows.

Assume the $2n+1$ numbers are split between two sites, $0 < m < 2n+1$ at one site and $2n+1-m$ at the other, and it is desired to compute the median using the least possible communications. It can be shown, using an argument similar to the above, that at least $b = \min(m, 2n+1-m)$ numbers must be transmitted. The minimum is achieved by sending all b of the numbers from the site with the smallest collection to the other site where the median will be computed. If the median is to be calculated at the smaller site, $b+1$ numbers must be transmitted. A similar analysis is available when the cardinality of the data set is even.

4 A Conjecture

A theorem and conjecture about Z -distributable functions are formed in terms of a partition induced on X by f . Let $f: X \times Y \rightarrow T$ and define

$$F_x = \{q \in X \mid \forall y \in Y : \langle f(q, y) = f(x, y) \rangle\}$$

$$\mathcal{F} = \{F_x \mid x \in X\}.$$

Thus, \mathcal{F} is a partition of X and the elements of an $F_x \in \mathcal{F}$ are indistinguishable from x vis-à-vis f .

Theorem 3 *If $f: X \times Y \rightarrow T$, $r: X \rightarrow W$ is continuous onto, where $W \subset X$ and $r(x) \in F_x$, and $p: W \rightarrow Z$ is bicontinuous 1-to-1 onto, then f is Z -distributable in X .*

Proof Let $d(x) = p(r(x))$ and $c(z, y) = f(p^{-1}(z), y)$. Then $f(x, y) = c(d(x), y)$ because the conditions guarantee that p^{-1} is well defined and continuous. \square

Conjecture 4 *If f is Z -distributable, then there is a $W \subset X$, a continuous onto $r: X \rightarrow W$, where $r(x) \in F_x$, and a bicontinuous 1-to-1 onto $p: W \rightarrow Z$.*

Figure 3 shows the relations among the sets and functions used to distribute f in the theorem and the conjecture. The key condition is that Z and W are topologically equivalent. The conjecture if true, together with the theorem, would provide an alternative, equivalent definition of Z -distributable functions.

It is straightforward to find W , r , and p that satisfy the conjecture for the computation of the mean and variance as described in Sections 2 and 3.

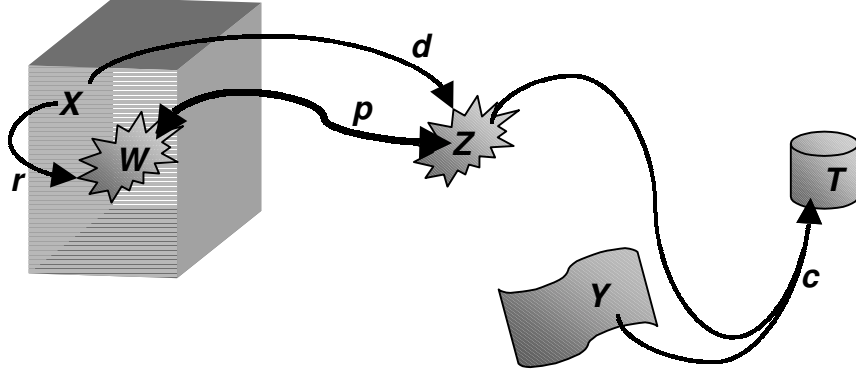


Figure 3: The conjecture is that Z -distribution by $c(d(x), y)$ entails a homomorphism p of Z and W , and $r: X \rightarrow W$ where $r(x) \in F_x$.

Let $s = (s_1, \dots, s_k)$ and $t = (t_1, \dots, t_k)$. Then, using the formulas shown in Figure 1, $t \in F_s$ if and only if $\sum t_i = \sum s_i$ and $\sum t_i^2 = \sum s_i^2$. Let

$$W = \{(a - \delta, a, \dots, a, a + \delta) \mid a \in \mathfrak{R} \wedge \delta \in \mathfrak{R}^+\}$$

and define $p: W \rightarrow Z$ as $p(w) = p(a - \delta, a, \dots, a, a + \delta) \rightarrow (a, \delta)$. Clearly, p is bicontinuous and 1-to-1 onto $Z = \mathfrak{R} \times \mathfrak{R}^+$ as required. It remains to find a continuous $r: X \rightarrow W$, onto, where $r(x) \in F_x$. Simply define $r(x_1, \dots, x_k) = (a - \delta, a, \dots, a, a + \delta)$, where $a = \alpha/k$, $\delta = \sqrt{(k\beta - \alpha^2)/(2k)}$, $\alpha = \sum x_i$, and $\beta = \sum x_i^2$. That $r(x) \in F_x$ is easily verified by substitution.

5 Discussion

In the theorem and conjecture, $W \subset X$ has a special significance—it provides a set of representatives for a partition of X that refines \mathcal{F} . Let

$$W_x = \{a \in X \mid r(a) = r(x)\}$$

$$\mathcal{W} = \{W_x \mid x \in X\}$$

\mathcal{W} clearly is a partition of X and it refines \mathcal{F} because

$$\forall x, a \in X : \langle W_x \subset F_a \vee W_x \cap F_a = \emptyset \rangle.$$

Essentially, p^{-1} maps Z into X such that its image, W , hits each element of \mathcal{W} once (and, hence each element of \mathcal{F} at least once) and r maps each $x \in X$ to a point in W that is also in F_x . The conjecture is that this is always possible if f is Z -distributable. In other words, if f is Z -distributable, the conjecture postulates a contraction, r , of X to a homomorphic image, W of Z , that preserves the partitioning induced on X by f , i.e., $r(x) \in F_x$.

In some cases it is possible to construct the W , r , and p , required by the conjecture from the d used to distribute f . If there is a 1-to-1 bicontinuous $q: Z \rightarrow X$ such that $d(q(z)) = z$, this is certainly the case: simply define $W = q(Z)$, $p(w) = q^{-1}(w)$, and $r(x) = q(d(x))$. Unfortunately, such a q does not in general exist for a given d (see Figure 4) so this is not a fruitful

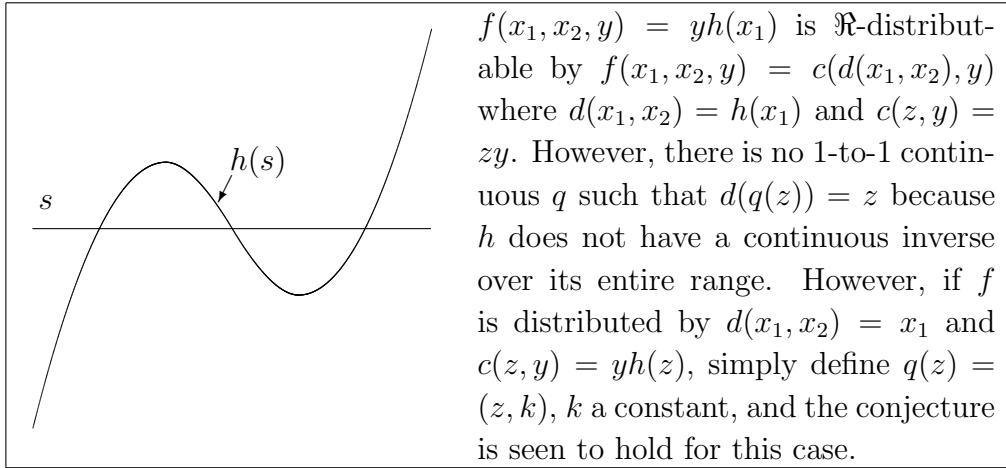


Figure 4: A problem with straightforward verification of the conjecture.

avenue to seek a general proof of the conjecture. However for some classes of functions, it is easy to show that the required q can be constructed from

any given d used to distribute the computation. An example is presented in the next section.

6 Distributing Monotonic Functions

Let $f: X \times Y \rightarrow \mathfrak{R}$ be Z -distributable, where $X = S^n$, $S \subset \mathfrak{R}$ is an interval, and $Z \subset \mathfrak{R}$. If $f(s_1, \dots, s_n, y)$ is strictly monotonic in each s_i , the q described in the previous section can be constructed as follows. Define W as the *main diagonal* of the hypercube, X , i.e.,

$$W = \{(\underbrace{s, \dots, s}_n) \mid s \in S\}.$$

Let d_W be the function d restricted to W . It will be shown that (1) d_W is 1-to-1 into Z and (2) d_W is onto Z . Hence, d_W^{-1} exists and is our q and, therefore, the conjecture will be established for this restricted class of monotonic functions. Consider

$$c(d(s, \dots, s), y) = f(s, \dots, s, y).$$

Since the right hand side is strictly monotonic in s , d_W must be strictly monotonic in s too and, hence, is 1-to-1 with a well-defined inverse from $d_W(W)$ to W . To show that d_W is onto Z , I will prove that for an arbitrary $x \in X$, there is a $w \in W$ such that $d(x) = d(w)$. Assume that f and d_W are increasing functions. If either or both are decreasing, virtually identical demonstrations are available.

Let $x = (s_1, \dots, s_n)$ be an arbitrary $x \in X$. If $x \in W$, there is nothing to show, so assume that $x \notin W$, i.e, not all of the s_i are equal, and define $m = \min s_i$ and $M = \max s_i$. Thus, there is at least one $s_i \neq m$ and one

$s_i \neq M$ so

$$f(m, \dots, m, y) < f(x, y) < f(M, \dots, M, y),$$

for all $y \in Y$, because of the strict monotonicity assumption. Therefore, if $d(m, \dots, m) \leq d(x) \leq d(M, \dots, M)$, the intermediate value theorem guarantees the existence of a $m \leq \beta \leq M$ such that $d(x) = d(\beta, \dots, \beta)$ and clearly $(\beta, \dots, \beta) \in W$.

The remaining cases are $d(x) < d(m, \dots, m)$ and $d(M, \dots, M) < d(x)$. Assume the latter—the demonstration for the first case is virtually identical. Define $g_i(t) = (1 - t)m + ts_i$ and $g(t) = (g_1(t), \dots, g_n(t))$. Note that $g(0) = (m, \dots, m)$, $g(1) = x$, and g is continuous. Note also that $f(g(t), y)$ is strictly increasing in t . Since $d(g(0)) < d(M, \dots, M) < d(x) = d(g(1))$, there must exist $0 < v < 1$ such that $d(g(v)) = d(M, \dots, M)$ by the intermediate value theorem. This in turn implies that $f(M, \dots, M, y) = f(g(v), y) < f(x, y)$. But this is a contradiction. Therefore, d_W is onto Z . The following theorem sums up this result.

Theorem 5 *The function $f: X \times Y \rightarrow \mathfrak{R}$, where $X = S^n$, $S \subset \mathfrak{R}$ is an interval, W is the main diagonal of X , and f is strongly monotonic in X , is Z -distributable, $Z \subset \mathfrak{R}$, if and only if there is a continuous onto $r: X \rightarrow W$, such that $r(x) \in F_x$.*

In other words, f is Z -distributable, $Z \subset \mathfrak{R}$, if and only if there is a continuous $\rho = \rho(x_1, \dots, x_n)$ such that $f(x_1, \dots, x_n, y) = f(\rho, \dots, \rho, y)$ for all $y \in Y$. This observation often provides a simple method to determine the distributability of a monotonic function. First consider a negative example:

$$f(x_1, x_2, y_1, y_2) = x_1 y_1 + x_2 y_2$$

defined for positive y_i . The idea is to find a $\rho = \rho(x_1, x_2)$ such that

$$f(x_1, x_2, y_1, y_2) = f(\rho, \rho, y_1, y_2).$$

But $\rho = (x_1 y_1 + x_2 y_2)/(y_1 + y_2)$ which necessarily depends on y_1 and y_2 . Therefore, this function cannot be Z -distributed, $Z \subset \mathfrak{R}$.

The same method provides a demonstration that the generalized means [1] are \mathfrak{R}^+ -distributable. These means are defined for each $v \in \mathfrak{R}$ as

$$h_v(x_1, \dots, x_n) = \lim_{z \rightarrow v} \left(\sum_{i=1}^n x_i^z / n \right)^{1/z},$$

where the x_i are positive. The limit operation is necessary for $v = 0, +\infty$, and $-\infty$ which correspond, respectively, to the geometric mean, maximum, and minimum. Consider distributing h_v where x_1, \dots, x_m , $m < n$, are the remote elements. To show that h_v is \mathfrak{R}^+ -distributable in these variables, it is only necessary to find a $\rho = \rho(x_1, \dots, x_m)$ such that

$$\begin{aligned} h_v(x_1, \dots, x_n) &= h_v(\rho, \dots, \rho, x_{m+1}, \dots, x_n) \\ \lim_{z \rightarrow v} \left(\sum_{i=1}^n x_i^z / n \right)^{1/z} &= \lim_{z \rightarrow v} \left(\left(\sum_{i=1}^m \rho^z + \sum_{i=m+1}^n x_i^z \right) / n \right)^{1/z}. \end{aligned}$$

Clearly there is a solution,

$$\rho = \lim_{z \rightarrow v} \left(\sum_{i=1}^m x_i^z / m \right)^{1/z},$$

that is a continuous function of x_1, \dots, x_m .

References

- [1] G. H. Hardy, J. E. Littlewood, and G. Pólya, *Inequalities*, Cambridge at the University Press, 1959.

Bibliography

Mr. Barnett studied undergraduate mathematics at Indiana University and UCLA and did his graduate work in Computer Science at the University of California, Irvine. He is currently a Principal Engineer at the Northrop Grumman Corporation, Automation Sciences Laboratory. His research interests include artificial intelligence, systems, and mathematics related to computation theory.

ENERGY EFFICIENT REDUNDANCY

JEFFREY A. BARNETT

1. SUMMARY

This note presents an alternative and simplified¹ formulation of a problem introduced by Dan Mosse and Rami Melham. The objective is to implement a triply redundant computation while minimizing energy expenditure. The strategy is to execute two of the three identical processes relatively rapidly. If the two processes agree on their results, all computation ceases. If they do not agree, the third process, executing at a slower rate, is sped up to complete its results by the deadline. The optimization result is shown to have the following form:

$$t_1 = f(p) \cdot t_2 \quad c_1 = g(p) \cdot c,$$

where p is the probability that processes 1 and 2 disagree, t_2 is the hard realtime deadline, $t_1 \leq t_2$ is the rendezvous time for processes 1 and 2, c is the number of cycles executed by each of the three processes, and $c_1 \leq c$ is the number of cycles process 3 executes by t_1 . Interestingly, $f(1/3) = g(1/3) = 1$ which means that full symmetric redundancy is as efficient as any seemingly more clever scheme when $p \geq 1/3$.

Section 2 presents the math models, Section 3 does the optimization, and Section 4 analyzes the results.

2. RESOURCE AND REDUNDANCY MODELS

Energy/Time Model:

- $t = c/v$, where t is the time to compute c cycles at voltage setting v .
- $e = cv$, where e is the energy expenditure to compute c cycles at voltage setting v .
- No energy is expended when no computation is being done.

Redundancy and Application Models:

- Three identical processes that each execute c cycles.
- Computation starts at time 0.

¹Read as less realistic.

- Error-free results, if available, must be provided by time t_2 , the hard deadline.
- Processes 1 and 2 will finish all c cycles by time $t_1 \leq t_2$.
- At time t_1 , process 3 will have completed execution of $0 \leq c_1 \leq c$ cycles.
- If processes 1 and 2 produce the same result at time t_1 , that result is used and all processing halts.
- If process 1 and 2 disagree, they halt but process 3 executes its remaining $c - c_1$ cycles in the period between t_1 and t_2 . Its results are combined/voted with the other results to determine the system's solution.
- Processes 1 and 2 fail to agree with probability p .

Figure 1 shows the time of execution of the three processes and the cycles executed in each period.

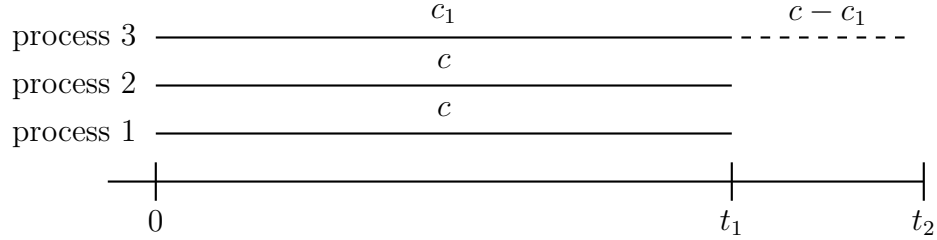


FIGURE 1. Time lines and cycles executed by periods.

3. OPTIMIZATION

Since $t = c/v$, $v = c/t$ is the voltage setting necessary to execute c cycles in time t . Since $e = cv$, $e = c^2/t$ is the energy expenditure necessary to execute c cycles in time t . Hence,

$$e_1 = \frac{c^2}{t_1} \quad e_2 = \frac{c^2}{t_1} \quad e_{31} = \frac{c_1^2}{t_1} \quad e_{32} = \frac{(c - c_1)^2}{t_2 - t_1},$$

where e_1 is the energy expenditure of process 1, e_2 is the energy expenditure of process 2, e_{31} is the energy expenditure of process 3 up to time t_1 , and e_{32} is the energy expenditure of process 3 between times t_1 and t_2 . Since the latter expense is only incurred, with probability p , if processes 1 and 2 disagree, the *expected* total energy expenditure is

$$\begin{aligned} E &= e_1 + e_2 + e_{31} + p \cdot e_{32} \\ &= 2\frac{c^2}{t_1} + \frac{c_1^2}{t_1} + p\frac{(c - c_1)^2}{t_2 - t_1}. \end{aligned}$$

The variables that control E are t_1 and c_1 ; c , t_2 , and p are constants. Thus, the straightforward approach to finding the t_1 and c_1 that minimize E is to simultaneously solve the equations $dE/dt_1 = 0$ and $dE/dc_1 = 0$. These equations are, after some rearrangement,

$$p \frac{(c - c_1)^2}{(t_2 - t_1)^2} = 2 \frac{c^2}{t_1^2} + \frac{c_1^2}{t_1^2} \quad p \frac{c - c_1}{t_2 - t_1} = \frac{c_1}{t_1}.$$

The simultaneous solutions are

$$t_1 = \frac{\sqrt{2pq} - 2q}{pq - 2q^2} \times t_2 \quad c_1 = \sqrt{\frac{2p}{q}} \times c,$$

where $q = 1 - p$. Note that $t_1 = t_2$ and $c_1 = c$ when $p = 1/3$. The expected energy utilization is

$$\begin{aligned} E &= \frac{(2\sqrt{q} + \sqrt{2p})(p - 2q)}{\sqrt{2p} - 2\sqrt{q}} \times \frac{c^2}{t_2} \\ &= \frac{(2\sqrt{q} + \sqrt{2p})^2}{2} \times \frac{c^2}{t_2} \end{aligned}$$

4. ANALYSIS

Figure 2 shows t_1 and c_1 as a function of p when nominally $c = 1$ and $t_2 = 1$. The figure also shows $E/3$ as a function of p . Its

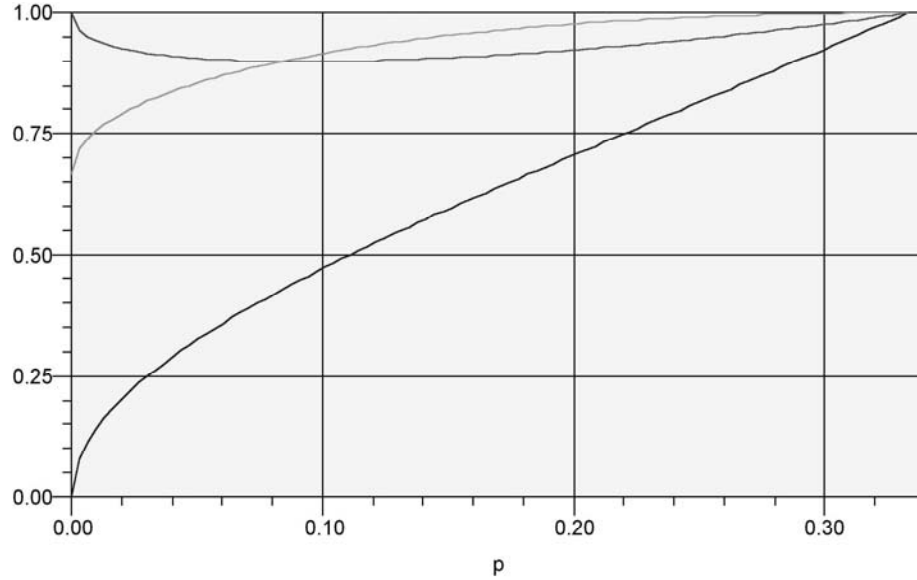


FIGURE 2. Plots of t_1 and c_1 versus p .

value monotonically increases from $2/3$ to 1 as p goes from 0 to $1/3$. Therefore, at most a 33% savings in energy is available when error-free hardware replaces the worst possible. Probably the most interesting thing to note is that the value of t_1 is always a sizable fraction of t_2 . When $p = 0$, $c_1 = 0$, i.e., process 3 never executes. As p nears $1/3$ the parameters make the execution of process 3 more and more like the other processes since there is a fair probability that its results will be needed.

POWER AWARE DUPLEX REDUNDANCY

JEFFREY A. BARNETT

This note addresses a problem posed and solved by Zhu, et al, in “Analysis of an Energy Efficient Optimized TMR.” The problem is how to chunk a process that will be executed in a duplex redundancy scheme where a portion of the process—a chunk—executes then compares progress with a companion executing on another host. If the processes agree on results to date, execution proceeds to the next chunk; otherwise, both processes rollback and reexecute the chunk, etc.

The question answered herein is how big should the chunks be in a minimum expected energy implementation where there is a hard real-time deadline. The model used is from Barnett’s “Minimum Energy Process Execution.”

For the nonce, assume the process is broken into n chunks, each of unit complexity. (Later, I’ll deal with the issues that chunk size depends on the overhead for bookkeeping and rollback.) Call the last chunk ‘1’ and the first chunk ‘ n ’. In Barnett’s model, the expected energy for each process is $E = s^2/t$, where t is the remaining time to deadline and s is a calculated parameter associated with chunk n . Our goal is to minimize s , hence expected energy, by choice of an appropriate n which will determine the chunk size.

First consider chunk 1, the last chunk executed, assumed of unit complexity. The s parameter associated with it is defined by

$$s_1 = 1 + \sqrt{p} s_1$$

where p is the error probability, i.e., with probability p , the chunk must be reexecuted; with probability $q = 1 - p$, there is a transition to the 0-complexity terminal state. Thus,

$$\begin{aligned} s_1 &= \frac{1}{1 - \sqrt{p}} \\ &= \frac{1 + \sqrt{p}}{1 - p} \\ &= \frac{1 + \sqrt{p}}{q}. \end{aligned} \tag{1}$$

For $n > i \geq 1$, the formula for s_{i+1} , by similar reasoning is

$$(2) \quad s_{i+1} = 1 + \sqrt{ps_{i+1}^2 + qs_i^2}$$

and, therefore,

$$(3) \quad s_{i+1} = \frac{1 + \sqrt{p + q^2 s_i^2}}{q}.$$

This form has resisted closed-form solution but it is easy to show that

$$(4) \quad \frac{n}{q} \leq s_n \leq \frac{n(1 + \sqrt{p})}{q}$$

from (1), (2), and (3) by induction. Since p is very small in practice, the bounds are fairly tight and, therefore, the approximation

$$s = s_n \approx n/q$$

is used below. It is now time to return to the chunk size question.

Assume that the total process comprises c cycles and that it is evenly divided into n chunks so that there are c/n process cycles in each chunk. In addition assume that z cycles are necessary to save or rollback state for each chunk execution. Therefore, the total chunk complexity is $z + c/n$ cycles and

$$(5) \quad \begin{aligned} s &\approx \frac{n}{q} \left(z + \frac{c}{n} \right) \\ &\approx \frac{nz + c}{q} \end{aligned}$$

for the whole process because the value of s scales linearly when the complexity of all chunks are multiplied by a constant. We assume that the probability of a failure while executing k cycles is $p = 1 - \exp(-\lambda k)$ so the probability of *no* failure for a chunk is

$$q = e^{-\lambda(z + \frac{c}{n})}.$$

Therefore, from (5) the quantity to minimize is

$$s \approx \frac{nz + c}{e^{-\lambda(z + \frac{c}{n})}}$$

by choosing n . Treating n as a real variable and solving $ds/dn = 0$ yields

$$(6) \quad \frac{c}{n} = \frac{2}{\lambda + \sqrt{\lambda^2 + 4\lambda/z}}$$

as the optimal complexity size of chunks, in cycles, to minimize expected energy utilization. This result is independent of c and t .

The bounds on expected energy $E = s^2/t$ for each of the duplex processes are computed from (4), (5), and (6) as

$$\begin{aligned} \frac{c^2(2 + \lambda z + \sqrt{\lambda^2 z^2 + 4\lambda z})^2}{4q^2 t} &\leq E \\ &\leq \frac{c^2(2 + \lambda z + \sqrt{\lambda^2 z^2 + 4\lambda z})^2}{4q^2 t} (1 + \sqrt{1 - q})^2, \end{aligned}$$

where

$$\begin{aligned} q &= \exp\left(-\lambda\left(z + \frac{c}{n}\right)\right) \\ &= \exp\left(-\lambda\left(z + \frac{2}{\lambda + \sqrt{\lambda^2 + 4\lambda/z}}\right)\right) \\ &= \exp\left(-\lambda z - \frac{2}{1 + \sqrt{1 + \frac{4}{\lambda z}}}\right). \end{aligned}$$

which isn't very informative.

NOTES

Zhu measures failure as a function of time; failure is measured as a function of cycles herein. I'm not sure which is a better model but the analysis was simplified in this note by the latter assumption. Zhu deals with the issues raised when not all voltages (execution speeds) are selectable; I do not. Further, Zhu uses a more sophisticated power model. The remaining work is to see if the quick and dirty approach used in this note concurs on the selection of chunk size. That comparison has not been performed. For small values of λ , Eq. (6) is approximated by

$$\frac{c}{n} \approx \sqrt{\frac{z}{\lambda}}$$

so, for example, if $z = 10^3$ and $\lambda = 10^{-9}$ (a rather pessimistic value), the best chunk size is approximately 10^6 cycles. If $m = \lambda^{-1}$ is substituted, the approximate optimal chunk size is \sqrt{mz} . And this is essentially proportional to the geometric mean of rollback complexity and the mean time between failures (MTBF).

MINIMUM ENERGY PROCESS EXECUTION

JEFFREY A. BARNETT

1. THE PROBLEM AND SUMMARY

A process that will execute on a single computer is given. The objective is to voltage schedule its subtasks so that process execution finishes before a hard deadline while minimizing the expected energy expenditure. The CPU voltage can be set for the execution of each subtask to tradeoff execution speed against energy.

The process is represented by a directed acyclic graph with a unique start node that is called a *conditional subtask graph*. Nodes represent subtasks and each node is annotated with the complexity of the subtask measured in CPU cycles. Directed edges connect a node to its possible execution successors. An edge is labelled with the probability that the subtask at the head of the edge will be selected for execution after the subtask at its tail. Exactly one task will be selected so this is a model of conditional execution of a single sequential (as opposed to a parallel) process. If a node has no successors, it is called a *terminal*. Exactly one terminal will be executed and it is required that its execution not exceed the hard deadline.

An algorithm to optimally voltage schedule a conditional subtask graph is developed. The algorithm processes the graph before execution and can be run offline. The time complexity is linear in the size of the graph. The time complexity at runtime to determine the optimal voltage for a subtask execution is $O(1)$.

If a transition probability is misestimated, the algorithm will automatically recover and schedule the remaining execution of the process optimally given the current state. Optimality recovery will also occur if subtask complexity estimates are inaccurate as long as an underestimate does not cause the deadline to be exceeded.

Section 2 describes the power model used throughout this note and discusses its fidelity. The mathematics in Section 3 determine the optimal voltage-scheduling policy for a conditional subtask graph. Formulas for the expected energy utilization for process continuations, starting at any subtask in the graph, are developed along with the

proper voltage setting for the subtask. Section 4 discusses the interpretation of a conditional subtask graph including conditional terminal nodes, nodes with multiple predecessors, and loops. Section 5 presents pseudo codes that implement the main algorithm and the API routine that adjusts voltage at the beginning of each subtask execution.

2. THE POWER MODEL

A simple first-order model is used to estimate the relations among task complexity, computation speed measured by delay, energy consumption, and voltage setting. The main assumptions are

- $t = c/v$, where t is the time necessary to execute c cycles at voltage setting v . Therefore, $v = c/t$ is the voltage setting necessary to execute c cycles in time t .
- $e = cv$, where e is the energy expenditure necessary to execute c cycles at voltage setting v . Therefore, $e = c^2/t$ is the energy necessary to execute c cycles in time t .
- The units of v and e are selected so that constants of proportionality can be omitted from the above.

This model has several sources of inaccuracies:

- It is assumed that all possible voltage settings are usable.
- It is assumed that the above relations hold even at low voltage settings where leakage current is a dominate effect.

Another assumption is that

- There is no energy overhead or delay associated with changing the voltage.

This source of inaccuracy can be partially mediated by increasing the estimates of the subtask complexities to include the expected overhead.

A final assumption is made, based on many published results for a variety of power models:

- The energy minimization strategy to execute a task of known fixed complexity in a given amount of time is to use the constant voltage setting that causes execution to exactly meet the given deadline, i.e., $v = c/t$, where c is the known complexity and t is the known deadline.

3. MATHEMATICAL DEVELOPMENT

Assume that a node of complexity c is about to be executed and the time remaining until the hard deadline is t . Assume that the node has n successors with the transition probabilities p_1, \dots, p_n . It will be shown that the optimal policy executes this subtask in time $t_1 \leq t$,

where $t_1 = ct/s$. Hence the proper voltage setting is $v = c/t_1 = s/t$. The *expected* total energy expenditure for executing this subtask and the remainder of the process is $E = s^2/t$. It remains to define s and prove this claim by induction.

The s in the above is defined to be $s = c + (\sum p_i s_i^2)^{1/2}$, where this and all other summations in this note run through the set of all direct successor nodes and s_i is the quantity, s , for the i^{th} successor node. If the node is a terminal, $s = c$ because there are no successors and the vacuous summation evaluates to zero. The quantity s is well defined since the graph has been assumed to be acyclic.

The induction proceeds in two steps: First, it is shown that the values of E and t_1 are optimal for terminal nodes. Second it is shown that the results are optimal for a nonterminal node given that it is optimal for its successor nodes.

Assume that a terminal node is to be executed so $s = c$. Thus, $t_1 = ct/s = t$ and $E = s^2/t_1 = c^2/t$ in agreement with the optimal just-in-time, constant-voltage energy minimizing policy to execute a node with known complexity in a fixed time period.

If the node is nonterminal, then time t_1 is found by which to finish the execution of the node. The remaining period, $t - t_1$, is used to execute the rest of the process. If t_1 were known, the energy expenditure at this node would be c^2/t_1 . If the i^{th} successor were selected to execute next, its expected energy consumption would be $s_i^2/(t - t_1)$ by the inductive assumption. Therefore, since the i^{th} successor is selected with probability p_i , the expected value of E , as a function of t_1 , is

$$(1) \quad E = \frac{c^2}{t_1} + p_1 \frac{s_1^2}{t - t_1} + \cdots + p_n \frac{s_n^2}{t - t_1}$$

The t_1 that minimizes E is found by solving $dE/dt_1 = 0$ for t_1 :

$$\begin{aligned} \frac{c^2}{t_1^2} &= \frac{\sum p_i s_i^2}{(t - t_1)^2} \\ ct &= (c + \sqrt{\sum p_i s_i^2}) t_1 \\ t_1 &= ct/s \end{aligned}$$

which was to be proved. It remains to show that $E = s^2/t$. That fact follows by substituting the optimal value of t_1 in (1). \square

4. TRICKS OF THE TRADE

This section discusses three topics: conditional terminal nodes, the fact that a given node can be the successor of multiple nodes, and modelling loops.

Consider a nonterminal node with n successors and transition probabilities p_1, \dots, p_n where $z = \sum p_i$ and $z < 1$. Such a node is called a *conditional terminal*. The interpretation is that executing the node terminates the process with probability $1 - z$. A careful inspection of the mathematics in the previous section shows that the results are equivalent to a formulation where an extra edge with transition probability $1 - z$ points to a terminal node with $c = 0$. Hence, the mathematics correctly deal with this case and interpretation. N.B., If a terminal node is executed by the process, its final cycle will coincide with the deadline. If a conditional terminal is the final subtask, the process will complete execution before the deadline.

The second issue is that a node may have multiple predecessors. In such cases, blind recursive evaluation of s for the start node could take time exponential in the size of the graph. The algorithm presented in the next section does much better, its time complexity is linear in graph size whether the graph is a tree or not.

The third issue is the representation of loops with conditional exits. Given the assumption of a hard deadline one can reasonably assume that there is a known finite iteration limit. The way to represent this case is to unroll the loop and consider each iteration as a separate subtask. Each subtasks except the final one has two successors:

- The subtask representing the next iteration of the loop.
- The subtask that follows loop termination.

Of course, transition probabilities must be estimated for both branches.

5. THE ALGORITHM

This section develops an efficient algorithm to voltage schedule a conditional subtask graph. The algorithm may be used offline. It calculates and stores the value of s for each node. An API function is called by each subtask at the beginning of its execution. The value of s calculated by the algorithm is passed as a parameter. The current time t and the deadline d are global variables. The API implementation is

```

procedure API_set_voltage (s number)
  if  $t \geq d$  then error("deadline missed");
  voltage  $\leftarrow s / (d - t)$ ;
end API_set_voltage

```

The error clause is only reached if a subtask complexity has been underestimated and its execution exceeds the deadline.

This simple algorithm works because the execution model is Markovian in the sense that the optimal strategy only depends on the subtask to execute next, its successors, and the remaining time budget, not how

we got here. Even if there were prior estimation errors, the continuation strategy is optimal from this point forward.

The calls to this API could most conveniently be generated by a power-aware compiler where the algorithm described below would reside. Of course the programmer or a statistics-gathering tool must estimate the transition probabilities.¹

The following algorithm calculates s for all nodes in the conditional subtask graph. Each node has fields that store c and a list of edges to the successors of the node. A third field, s , is initially set to a negative number by the graph builder. The s field has two uses:

- As a marker—when the value is negative the node has not been processed.
- To store s —when its value is nonnegative, it contains the correct value of s for the node.

The use of this field as a mark bit is what keeps the time complexity of the algorithm proportional to graph size even if some nodes have multiple predecessors. The successors of a node are only chased on the first visit to the node. Each edge structure stores its transition probability and the node at its head.

```

number function comp_s (node node)
  if node.s < 0 then {
    declare x number;
    x ← 0;
    for edge in node.edges {
      x ← x + edge.prob × comp_s(edge.head)2};
    node.s ← node.c +  $\sqrt{x}$ };
  return node.s;
end comp_s

```

This algorithm is invoked by evaluating $\text{comp_s}(\text{start})$, where start is the unique start node of the conditional subtask graph. When it is complete, every reachable node contains its correct s value. These are the values that will be passed to the calls on API_set_voltage at the beginning of each subtask execution.

¹The old FORTRAN frequency statement may have a new life!

Dynamic Voltage Scheduling Optimizations

Jeffrey A. Barnett

Abstract

Energy versus delay tradeoff are explored for systems that must manage energy expenditure as well as computation deadlines. The focus is execution of a single process on a single processor. Two probabilistic process models are considered along with a family of power dissipation models. The first process model assumes that process complexity is exactly c cycles with probability $p(c)$. The second model considers the detailed branching and loop structure of the code. Probabilities are attached at branch points. The power models assume that energy dissipation per cycle is proportional to v^m and that execution time for a cycle is proportional to v^{-n} , where v is supply voltage. The energy versus delay tradeoff is implemented using dynamic voltage and clock adjustments. The problems solved include (1) minimize expected execution time given a hard energy budget and (2) minimize expected energy expenditure given a hard deadline. The problem of minimizing the expected value of $Q(E, T)$, where Q is a penalty function and E and T are, respectively, total energy and total time, is also solved using the first process model. Analysis determines theoretical conditions where it may be profitable to switch voltage or modify an a priori voltage schedule.

Keywords: Energy-aware systems, energy management, time management, dynamic voltage scheduling, agile voltage scheduling, power management points.

Acknowledgements: This effort is sponsored by Defense Advanced Research Projects Agency (DARPA) through the Air Force Research Laboratory, USAF, under agreement number F33615-02-C-4001.

1 Background, Summary, and Contributions

1.1 Background

Resource management has always been a primary focus area within computer science. Virtually all pragmatic models of computation contain a layer—the operating system—dedicated to it. The resources managed typically include the CPU, memory, secondary storage, and device access. Object controllers and other systems that interact with their environments have special resource-management constraints that are called hard (or soft) realtime requirements. Properties of, and algorithms for realtime systems and resource management are among the most studied in computer science.

Energy is a relatively recent addition to the list of resources that must be managed. This is driven, in part, by increased computation capabilities and miniaturization that entail heat density and dissipation problems [36]. Thus, energy management problems exist for large systems such as servers [28, 35, 36] as well as portable devices [11, 27, 28, 35]. Another driver is economics: energy costs money.

There are other classes of systems that must be radical more power aware. Two examples are space systems [40] and unattended, distributed sensor networks [45]. The Sky Tower aircraft with an endurance goal of six months using electric engines [2] is a third example. These systems use batteries as buffers but must scavenge energy from the environment to achieve long endurance. Solar panels are one potential energy source; vibration, chemical, and ambient heat are other possibilities. Energy availability is not constant so optional activities must be scheduled for times of plenty, e.g., when solar panels are illuminated, and all but the most urgent needs must be deferred at other times.

Several approaches to energy management are currently being pursued. Many researchers are investigating hardware improvements that will reduce energy consumption with minimal or no impact on runtime performance. Examples include use of two different supply voltages within the same chip [18], use of two identical processors where each has a different supply voltage [43], dynamical control of L2 cache line size [23], distribution within a super scalar chip design [47], use of FPGA devices [31], and design of bus structures that use less energy [26]. These approaches promise “automatic” savings because the hardware is more energy efficient. Several studies [23, 29, 47] propose metrics to measure the ultimate success of these investigations. Proposed minimization objectives have the form ET^α , where E is

the energy used by a test computation, T is its execution time, and α is a constant.

Another approach—the one investigated herein—provides controls used by the application and/or the operating system to effect an energy versus performance tradeoff at runtime. This possibility is interesting because many, if not most, energy-constrained systems also have temporal constraints [3, 30]. The available controls are the supply voltage and the clock frequency. When the voltage is increased, the clock frequency can be increased. However, the energy cost per unit of computation will also increase. Several commercially available processors provide voltage/clock controls: Intel Xscale [19], Transmeta Crusoe [42], AMD K6-2+[1], and the IBM StrongArm [21] are some examples. The Advanced Configuration and Power Management (ACPI) specification provides standard interfaces for low-power states and energy and thermal controls as well as plug-and-play hardware protocols [20].

Generally, researchers following the program pursued herein concentrate on minimizing energy consumption given hard or soft temporal constraints [3, 11, 15, 27, 28, 30, 35, 36, 39, 43, 46]. Of these, only Cao [11] and Qiu and Pedram [36] note that the dual problem—minimize expected execution time given an energy budget—is also significant. Both formulations are addressed below.

1.2 Summary

Section 2 introduces the power models and the process models used for algorithm development and analysis. The power models specify relations between supply voltage levels and processor speed and rate of energy consumption. The process models encode probabilistic knowledge about process complexity measures in cycles. Two process models are examined: The *simple probabilistic model* provides the function p , where $p(x)$ is the probability that process complexity is exactly x cycles. A *structural process model* provides a flow graph of the process. The associated metrics are the complexities of the simple segments (the graph nodes) and the probabilities of branching from one segment to another one.

Section 3 formulates a set of optimization problems and solves them analytically using a specific simple power model, then Section 4 provides the optimal solutions for the same optimizations using arbitrary power models. Two optimizations—find the voltage schedule that minimizes expected execution time given a not-to-exceed energy budget and find the voltage schedule that minimizes expected energy consumption given a not-to-exceed deadline—are solved for both the simple probabilistic and structural power models. A third optimization—

find the voltage schedule that minimizes the expected value of $Q(E, T)$, where E is total energy expenditure, T is total computation time, and Q is a general penalty function—is solved for the simple probabilistic process model.

Section 5 discusses methods for gathering process metrics—the probabilities and complexity estimates that comprise process models—and how to insert that information in application systems. The probabilities are available from general application knowledge, domain models, and feedback from system executions. The complexity measures, on the other hand, are generated from machine models, cycle-level simulators, and compiler analyses.

Section 6 analyzes the optimality results. Optimal strategies are compared to strategies that predict average-case behavior. The optimal strategies always behave as though future complexity will be worse than average. Other analyses discuss the value of using updated complexity information promptly and the robustness of the optimal algorithms to parameter estimation errors. Finally, it is shown that voltage levels in optimal strategies can only change at points where something new is learned about future complexity of the process. At all other points, voltage remains constant.

1.3 Contributions

The focus of this article is how best to use voltage and clock controls to effect application-driven energy versus delay tradeoffs. The investigation assumes a single process or a dependent set of tasks executing on a single processor. This restriction is imposed for two reasons: The first is that many energy-constrained systems are organized this way [39]. The second reason is that exact, basic optimization methods are needed as the building blocks for schedulers that handle multiple processes executing on multiple processors.

Several results developed below appear to be novel. The first is the solution of optimization problems where energy is the constrained resource and expected execution time is to be minimized. The second is the formulation and solution to minimizing the expected value of the general penalty function $Q(E, T)$, where E is energy consumption and T is execution time. The third contribution is the development of optimal techniques to schedule processes from their structural models. Perhaps the most significant result is an analysis that determines when an optimal voltage schedule will change voltage and hence computation speed and energy consumption rate, and when these parameters will remain constant: voltage level will only change in an optimal schedule when something *new* is learned about the future

complexity of the process.

2 Process and Power Models

Section 2.1 introduces the power models and Section 2.2 introduces the process models used for the optimizations performed in Sections 3 and 4.

2.1 Power Models

Many modern computers permit voltage and clock frequency to be modified by the executing program. Some examples are the Intel Xscale [19], the Transmeta Crusoe [42], the AMD K6-2+ [34], and the IBM StrongArm [37]. Increased voltage permits increased clock frequencies to be used. The penalty for faster computations is increased energy consumption per unit of computation. That unit is the cycle. Cycles are used to measure execution progress rather than instructions because cycles are reasonably similar to one another in terms of energy dissipation and execution time while instructions are not [27, 30, 39, 46].

The general models for the energy expenditure and time to execute c cycles, respectively, are $e = \alpha c v_{dd}^m$ and $t = \beta c v_{dd} / (v_{dd} - v_T)^{n+1}$, where v_{dd} is supply voltage (the variable) and v_T is threshold voltage [9, 13, 38]. The constants v_T , α , β , m , and n depend on the architecture. Martin [29] discusses “smooth circuits” where v_T scales with v_{dd} so that $t = \beta' c / v_{dd}^n$ would represent execution time, and many authors [15, 27, 35, 39, 43], while acknowledging the effects of v_T , ignore it either in optimizations, examples, or testing. Manzak and Chakrabarti [28] note that when $v_{dd} > 3v_T$, only a 0.1% error is introduced if $v_T = 0$ is assumed, i.e., the voltage schedule developed ignoring v_T uses only 0.1% more energy than an optimal schedule. For these reasons, the power model can be simplified to $e = \alpha c v^m$ and $t = \beta c / v^n$, where $v = v_{dd}$. A further notational simplification follows if the *units* of e and t are chosen to entail $e = c v^m$ and $t = c / v^n$.

Two limitations on speed adjustments should be noted: (1) voltage and clock levels are discrete and (2) minimum and maximum levels are imposed by the hardware. For example, the Transmeta Crusoe [42] provides as many as two clock frequencies for a single voltage level and 36 combinations. However, Gutnik and Chandrakasan [16] and Namgoang et al [33] discuss architectures where continuous speed adjustments are possible and Melham et al [30] note that systems which are able to operate on a (more or less) continuous voltage

spectrum are rapidly becoming a reality thanks to advances in power supply engineering and CPU design. Chandrakasan et al [12] shows that a few voltage/speed levels are sufficient to achieve almost the same energy savings as infinite levels.

The development below follows Lorch and Smith [27] and Shin et al [39] by assuming that voltage can vary continuously. However, many studies [15, 30, 46] do deal with issues of discrete and limited settings and their work should be consulted when these effects are important in particular applications. Techniques to deal with limitations on minimum and maximum voltage settings are discussed elsewhere [3, 46].

It should also be noted that changing processor speed through voltage and/or clock adjustments may exact time and energy penalties [10, 19, 34, 37, 42, 46]. Lorch and Smith [27] assume that changing voltage incurs little or no overhead and Aydin et al [3] suggest, as will be assumed here, that these penalties simply be added to the complexity estimates of process segments and otherwise be ignored during optimization.

The power models appearing in the optimizations developed below are denoted by Π_{mn} . Using the Π_{mn} model, the energy to execute c cycles at constant voltage v is $e = cv^m$ and the execution time is $t = c/v^n$. Some optimizations will assume that voltage is *agile*, i.e., voltage can vary continuously. In such cases,

$$e = \int_0^c v(x)^m dx \quad t = \int_0^c v(x)^{-n} dx,$$

where $v(x)$ is the voltage used when cycle x is executed. The Π_{21} model is the one most often used [3, 15, 28, 35, 43, 46]. Lorch and Smith [27] also do basic analysis with Π_{21} but note that formulas developed using Π_{11} better fit simulation results. Shin et al [39] also develop optimizations using Π_{21} but evaluate an example architecture with a $\Pi_{2(.7)}$ model. All optimizations below are initially performed with the Π_{11} model in order to simplify derivations. Section 4 re-presents the results using general Π_{mn} models.

Melham et al [30] proves, for a variety of models, that the lowest energy utilization to execute a fixed number of cycles when there is a hard deadline is achieved by the constant voltage solution that uses all of the available time. Isihara and Yasuura [22] note the same result and Błazewicz et al [6] note that when the rate of consumption of some resource is a convex function of CPU speed, an ideal schedule will run each task at a constant speed. A corollary is used below: the least time to execute a fixed number of cycles when there is a hard energy constraint is achieved by the constant voltage solution that uses all of the

available energy.

2.2 Process Models

The unit presented for scheduling is the process. Two types of process models are defined below. The first is the simple probabilistic model where all that is known about a process is that its complexity is x with probability $p(x)$. The second is the structural model where the branching behavior and probabilities of the branches are specified. The second model is clearly more informed than the first one—a simple probabilistic model can be derived from structural information though there is an information loss—so better optimizations are possible.

2.2.1 The Simple Probabilistic Process Model

The simple probabilistic process model is specified by a nonnegative function, p , where $p(x)$ is the probability that execution will terminate after exactly x cycles. Two assumptions are made: (1) $p(x) = 0$ if $x \leq 0$ and (2) there exists a finite c such that $p(x) = 0$ if $x > c$. In the notation used below, c will always denote the *smallest* value for which $\int_0^c p(x) dx = 1$.

The function $P(x) = \int_0^{x-} p(y) dy$ is the probability that process complexity is less than x and $z(x) = \int_x^c p(y) dy$ is the probability that the complexity is at least x . Thus, $z(x) = 1 - P(x)$. A simple probabilistic process model will often be specified by P rather than p . Though the information content is equivalent, the former is often more straightforward to elicit and represent.

Both [15] and [27] introduce simple probabilistic models and find voltage schedules that minimize expected energy consumption given a not-to-exceed deadline. The latter uses their results to improve performance of an existing dispatch schedule where the deadlines are already defined.

2.2.2 The Structural Process Model

The structural model of a process provides details of its branching. A process is represented by a 4-tuple, (σ, S, c_x, p) , where S is a set of code segments, $c_x(s)$ is the complexity measured in cycles of each $s \in S$, and σ is the initial segment (entry point) of the process. Branch probabilities are represented by p , where $p(s_1, s_2)$ is the probability that execution of s_2 will

immediately follow execution of s_1 . Let $p(s) = \sum_{r \in S} p(s, r)$, then clearly $p(s) \leq 1$ is required for all $s \in S$. If $p(s) < 1$, then $1 - p(s)$ is the probability that an execution of s will be process terminal. Define $\theta(s)$ to be the set of all $r \in S$ such that $p(s, r) > 0$.

A directed graph can be used to represent a structural process model. The $s \in S$ are the nodes of the graph and there is a directed edge from each $s \in S$ to the members of $\theta(s)$. The nodes are labeled by $c_x(s)$ and the arc from s to $r \in \theta(s)$ is labeled with $p(s, r)$. For the nonce, assume that process graphs are acyclic. Methods that deal with graph cycles will be discussed in Section 3.3 where loops are analyzed.

Many investigators use flow-graph technology to represent processes that will be voltage scheduled to minimize energy consumption given computational deadlines. The simplest case is a graph where the edges represent simple order dependencies [43]. Both [30] and [39] use graph models where probabilities annotated branches and worst- and average-case complexities are calculated for the continuations starting at each node. Zhu et al [46] present a generalize of an AND/OR model [14] where AND nodes represent parallel dispatch, OR nodes, where synchronization is forced, represent conditional branching, and probabilities annotate these branches. Worst- and average-case complexity statistics are used to improve voltage schedules within an innovative slack-stealing scheme.

2.2.3 Process Model Comparison

Consider a process with three segments. The first segment, with $c_x = 15$, is executed then there is a branch to one of two final segments: one with $c_x = 10$ and the other with $c_x = 20$. Figure 1(a) shows the structural model of this process with the assumption that both branch

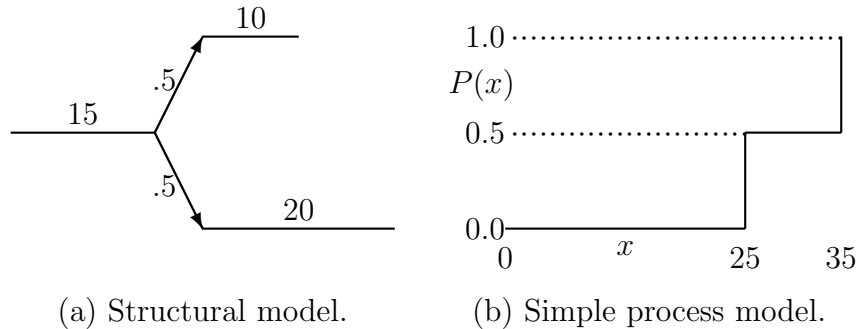


Figure 1: The structural and simple models for the same process.

probabilities are 0.5. Figure 1(b) shows $P(x)$ for the simple probabilistic model of the same

process. In the simple model $p(25) = .5$, $p(35) = .5$, and all other p -values are zero. The relevant observation is that the structural model is more informed: total process complexity is known in the structural model at cycle 15 when the branch is selected. The simple model only provides new complexity information at cycle 25 where the process either halts or continues for 10 more cycles. As will be shown below, the delay in knowledge capture increases the expected energy and time measures that we are trying to minimize.

3 Optimizations

Optimized voltage schedules are derived using simple probabilistic and structural process models assuming the Π_{11} power model. The optimization results are repeated with the general Π_{mn} model in Section 4. Table 1 summarizes the optimizations performed in this

Table 1: Section 3 optimization summary.

Section	Model	Budget	Minimize
3.1.1	Simple	E	$\text{Ex}(T)$
3.1.2	Simple	T	$\text{Ex}(E)$
3.1.4	Simple	—	$\text{Ex}(Q(E, T))$
3.2.1	Structural	E	$\text{Ex}(T)$
3.2.2	Structural	T	$\text{Ex}(E)$

section, where $\text{Ex}(\cdot)$ is the expected value operator, E is energy expenditure, and T is computation time. Sections 3.1.3 and 3.1.5 provide optimization examples, Section 3.2.3 describes algorithms to implement optimizations with structural models, and Section 3.3 discusses the modeling and handling of loops, particularly when the number of iterations is not bounded at compile time.

3.1 Simple Probabilistic Model Optimizations

3.1.1 Hard Energy Bound

The problem is to find the agile voltage schedule that minimizes the expected execution time of a process given its simple probabilistic model, p , and a not-to-exceed energy budget E . The expected execution time is $\int_0^c z(x)/v(x) dx$ because $z(x)$ is the probability that the

process will still be executing at cycle x and $v(x)^{-1}$ measures the time to execute that cycle. The energy constrain is $E = \int_0^c v(x) dx$. This problem is solved using a Lagrangian multiplier and variational methods [8]. The objective has the form,

$$\int_0^c \frac{z(x)}{v(x)} dx + \lambda \left[E - \int_0^c v(x) dx \right],$$

where λ is the Lagrangian multiplier. Substitute $v(x) + \beta g(x)$ for $v(x)$, differentiate with respect to β , let $\beta = 0$, and set the result to zero:

$$\begin{aligned} - \int_0^c \frac{g(x)z(x)}{v(x)^2} dx - \lambda \int_0^c g(x) dx &= 0 \\ \int_0^c g(x) \left[\frac{z(x)}{v(x)^2} + \lambda \right] dx &= 0. \end{aligned}$$

Since $g(x)$ can be an arbitrary function, $\lambda + z(x)/v(x)^2 = 0$ is necessary. Therefore, $v(x) = kz(x)^{1/2}$ for some constant k . From $E = \int_0^c v(x) dx$, it follows that $k = E/I$, where $I = \int_0^c z(x)^{1/2} dx$. Thus, a summary of the minimization result is

$$v(x) = \frac{E}{I} z(x)^{1/2} \quad \text{Ex}(T) = \frac{I^2}{E} \quad \text{Ex}(E) = \frac{E}{I} \int_0^c z(x)^{3/2} dx, \quad (1)$$

where $\text{Ex}(T) = \int_0^c z(x)/v(x) dx$ is the expected process execution time as noted above and $\text{Ex}(E) = \int_0^c z(x)v(x) dx$ is expected energy utilization. Note that $\text{Ex}(E) \leq E$ because $0 \leq z(x) \leq 1$.

3.1.2 Hard Time Bound

The problem is to find the agile voltage schedule that minimizes the expected energy utilization of a process given its simple probabilistic model, p , and a not-to-exceed time budget T . The method to solve this problem is identical to the one used in Section 3.1.1. The summary of this minimization is

$$v(x) = \frac{I}{Tz(x)^{1/2}} \quad \text{Ex}(T) = \frac{T}{I} \int_0^c z(x)^{3/2} dx \quad \text{Ex}(E) = \frac{I^2}{T}. \quad (2)$$

This result, for different power models, also appears in [15] and [27].

3.1.3 Examples for Hard Energy and Time Bounds

Let $p(x) = \frac{\pi}{2c} \sin(\frac{\pi}{c}x)$ when $0 \leq x \leq c$ and $p(x) = 0$ elsewhere be a simple probabilistic model. In the first problem, E is a hard energy bound and in the second, T is a hard time bound.

Table 2: Optimal solution for hard energy and time bounds.

Budget	$v(x)$	$\text{Ex}(E)$	$\text{Ex}(T)$
E	$\frac{\pi E \cos\left(\frac{\pi}{2c}x\right)}{2c}$	$\frac{2}{3}E$	$\frac{4c^2}{\pi^2 E}$
T	$\frac{2c}{\pi T \cos\left(\frac{\pi}{2c}x\right)}$	$\frac{4c^2}{\pi^2 T}$	$\frac{2}{3}T$

Table 2 presents the optimal agile voltage solutions and expected resource consumptions for both problems. It is interesting to note that voltage is a decreasing function when energy is constrained and an increasing function when time is constrained. This will always be the case as an examination of (1) and (2) will reveal. The rising effect was previously noted [27] and led to an approach named Processor Acceleration to Conserve Energy (PACE).

3.1.4 General Penalty Function

The problem is to find the agile voltage schedule that minimizes the expected value of $Q(E, T)$, where Q is a penalty function and E and T are, respectively, total energy consumption and total execution time, given a simple probabilistic model, p . The objective to minimize is, thus,

$$\text{Ex}(Q(E, T)) = \int_0^c p(x) Q(E(x), T(x)) dx,$$

where $E(x) = \int_0^x v(y) dy$ and $T(x) = \int_0^x v(y)^{-1} dy$. This problem is tackled by variational methods [8] starting with the substitution $v(x) + \beta g(x)$ for $v(x)$, differentiating with respect to β , letting $\beta = 0$, and equating the result to zero:

$$\int_0^c p(x) \left[Q_1 \int_0^x g(y) dy - Q_2 \int_0^x \frac{g(y)}{v(y)^2} dx \right] dx = 0,$$

where $Q_1 = \partial Q / \partial E$ and $Q_2 = \partial Q / \partial T$. Change the integration order to produce the equivalent

$$\int_0^c g(x) dx \int_x^c p(y) \left[Q_1 - \frac{Q_2}{v(x)^2} \right] dy = 0.$$

Therefore, minimizing/stationary voltage schedules must satisfy

$$v(x)^2 = \frac{\int_x^c p(y) Q_2 dy}{\int_x^c p(y) Q_1 dy} \quad (3)$$

since g can be arbitrarily chosen.

Integral equations such as (3) are notoriously difficult to solve. However, some cases are straightforward. For example, there is often a constant v that satisfies (3) if $Q_2/Q_1 = f(T/E)$ for a suitably well-behaved f and Π_{11} is assumed. Let v be a constant, then

$$\begin{aligned} f(T/E) &= f\left(\frac{\int_0^x v \, dy}{\int_0^x v^{-1} \, dy}\right) \\ &= f(v^2), \end{aligned}$$

which is a constant. Therefore, $Q_2/Q_1 = f(v^2)$ and, from (3),

$$\begin{aligned} v^2 &= \frac{\int_x^c p(y) f(v^2) Q_1 \, dy}{\int_x^c p(y) Q_1 \, dy} \\ &= f(v^2). \end{aligned}$$

So the solutions of $v^2 = f(v^2)$, i.e., the square root of the fixed points of f , if any, are stationary solutions of (3). These conditions are met whenever $Q(E, T) = g(ET)$ or $Q(E, T) = g(\alpha E^r + \beta T^r)$ as simple computations will show.

3.1.5 General Penalty Function Example

Let $p(x) = c^{-1}$ when $0 \leq x \leq c$ and let $p(x) = 0$ elsewhere. Then p is a simple probabilistic process model. Let the penalty function Q have the form $Q(E, T) = E + \frac{3}{2}c^2(1 - e^{-2T})$ so that $Q_1 = 1$ and $Q_2 = 3c^2e^{-2T}$. Therefore, a stationary solution must satisfy

$$\begin{aligned} v(x)^2 &= \frac{\int_x^c p(y) Q_2 \, dy}{\int_x^c p(y) Q_1 \, dy} \\ &= \frac{\int_x^c 3ce^{-2T} \, dy}{\int_x^c \frac{1}{c} \, dy} \\ &= \frac{3c^2 \int_x^c e^{-2T} \, dy}{c - x}. \end{aligned}$$

The above is satisfied by $v(x) = c - x$, where $T(y) = \log \frac{c}{c-y}$, as can be verified by substitution.

3.2 Structural Process Model Optimizations

3.2.1 Hard Energy Bound

The problem is to find a voltage schedule that minimizes expected execution time given the structural process model, (σ, S, c_x, p) , and a not-to-exceed energy budget E . Define the

quantity

$$I(s) = c_x(s) + \sqrt{\sum_{r \in \theta(s)} p(s, r) I(r)^2} \quad (4)$$

for each $s \in S$. Note that $I(s) = c_x(s)$ when s is a terminal segment because $\theta(s) = \emptyset$ and, hence, the summation is vacuous.

It is shown here that the optimal voltage, $v(s)$, to execute segment s and the expected execution time for s and the remainder of the process are

$$v(s) = \frac{e}{I(s)} \quad \text{Ex}(t(s)) = \frac{I(s)^2}{e}, \quad (5)$$

where e is the remaining energy budget when s is executed. Thus, the minimum expected execution time for the whole process, while honoring the energy budget, is $I(\sigma)^2/E$.

The first thing to note is that each terminal segment, s , will execute with a constant voltage because the number of cycles $c_x(s)$ and the energy budget allocation are fixed. (See Section 2.1.) From $e = c_x(s)v$, it follows that $v = e/c_x(s) = e/I(s)$ and from $t = c_x(s)/v$ it follows that $t = I(s)^2/e$; both in agreement with (5). This completes the base step for an induction to verify (5).

Now consider a segment s where (5) describes the optimal policy for all $r \in \theta(s)$. Let e be the remaining budget energy budget. Some of that budget, e_1 , will be devoted to s and the remainder, $e - e_1$, will be used for the rest of the process execution. Thus, the expected time to execute s plus the remainder of the process is

$$\begin{aligned} \text{Ex}(t(s)) &= \frac{c_x(s)^2}{e_1} + \sum_{r \in \theta(s)} p(s, r) \text{Ex}(t(r)) \\ &= \frac{c_x(s)^2}{e_1} + \sum_{r \in \theta(s)} p(s, r) \frac{I(r)^2}{e - e_1} \\ &= \frac{c_x(s)^2}{e_1} + \frac{(I(s) - c_x(s))^2}{e - e_1} \end{aligned} \quad (6)$$

using the inductive assumption and a constant-voltage solution for s . The minimizing value of e_1 is found by solving $\frac{d \text{Ex}(t(s))}{d e_1} = 0$ for e_1 . The result is $e_1 = c_x(s)e/I(s)$. Since $e_1 = c_x(s)v(s)$, it follows that $v(s) = e/I(s)$ in agreement with (5). Now substitute the optimizing value of e_1 into (6) and simplify to show that $\text{Ex}(t(s)) = I(s)^2/e$ and complete the induction.

3.2.2 Hard Time Bound

The problem is to find a voltage schedule that minimizes expected energy utilization given the structural process model, (σ, S, c_x, p) , and a not-to-exceed time budget T . The method

to solve this problem is identical to the one used in Section 3.2.1. The results of this optimization are

$$v(s) = \frac{I(s)}{t} \quad \text{Ex}(e(s)) = \frac{I(s)^2}{t}, \quad (7)$$

where $I(s)$ is defined by (4) and t is the remaining time budget when s is executed. The expected energy utilization for the entire process is $I(\sigma)^2/T$.

3.2.3 Algorithms for Structural Models

The optimal policies developed for the structural process model are straightforward to implement. Voltage is set when each $s \in S$ begins execution as a function of $I(s)$ and the remainder of the budgeted resource; how control arrives at s is not relevant. Figure 2 shows API routines that are called at the beginning of the execution of s with the parameter $I(s)$.

<pre> <i>procedure</i> set_voltage (I) <i>if</i> $e \leq 0$ <i>then error</i>; voltage $\leftarrow e/I$; <i>end set_voltage</i> </pre>	<pre> <i>procedure</i> set_voltage (I) <i>if</i> $t \leq 0$ <i>then error</i>; voltage $\leftarrow I/t$; <i>end set_voltage</i> </pre>
(a) Energy management API.	(b) Time management API.

Figure 2: APIs to support structural process model execution.

The API shown in Figure 2(a) is used when energy is budgeted and e is the remaining energy budget. Figure 2(b) shows the API when time is the constraint and t measures remaining time. The error checks account for possible prior effects of parameter misestimations.

In order to use the optimal policy, $I(s)$ for each $s \in S$ must be available at runtime. These values can be calculate offline and inserted, along with the API calls, by a power-aware compiler. The total execution time to calculate all $I(s)$ values is proportional to the size of the graph used to represent the structural model of the process as an examination of Figure 3 will show. The recursive algorithm is called with σ , the initial segment, and when it completes, $\mathbf{s}.I$ is set for each $s \in S$. The mark fields $\mathbf{s}.mark$ are initially **false**; they are used so that $\mathbf{s}.I$ is calculated exactly once for each $s \in S$.

```

function I(s segment)
  declare a number;
  if  $\neg$ s.mark then {
    s.mark  $\leftarrow$  true;
    a  $\leftarrow$  0;
    for  $r \in \theta(s)$  {
      a  $\leftarrow$  a + prob( $s, r$ ) * I( $r$ )2};
    s.I  $\leftarrow$   $c_x(s) + \sqrt{a}$ };
  return s.I;
end I

```

Figure 3: Algorithm to calculate $I(s)$ for each $s \in S$.

3.3 Loops

Loop constructs are a fundamental control structure that must be accounted for by any serious process modeling technique. The possibility of variable numbers of iterations is one of the major contributors to nondeterminacy in process complexity. The simple probabilistic model captures and organizes this information when p is estimated. That model is certainly the easiest and most natural one to use when there are many loops with variable iteration counts or elaborate control structures composed of ill-nested forms. Since many systems that interact with the environment have such characteristics, their architects will often choose the simple probabilistic model.

The structural process model also provides mechanisms to account for loops. If, for example, the iteration count is fixed, the loop is simply unrolled. If only the maximum count is known, the loop is still unrolled with appropriate probabilities attached to the continuation and exit branches of each iteration.

The case where the iteration limit is unknown in advance or is difficult to calculate is more problematic. Consider the canonical loop structure shown in Figure 4 where segment h is the head of the loop and segment b is the body. The header enters the body with probability p and segment r , the loop exit, with probability $q = 1 - p$. Two equations immediately follow from (4):

$$I(h) = c_x(h) + \sqrt{pI(b)^2 + qI(r)^2}$$

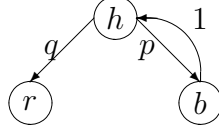


Figure 4: Simple loop structure.

$$I(b) = c_x(b) + I(h).$$

The simultaneous solutions for $I(h)$ and $I(b)$, with the assumption that p is independent of iteration count, are

$$I(h) = \frac{c_x(h) + pc_x(b) + \sqrt{p(c_x(h) + c_x(b))^2 + q^2 I(r)^2}}{q}$$

$$I(b) = \frac{c_x(h) + c_x(b) + \sqrt{p(c_x(h) + c_x(b))^2 + q^2 I(r)^2}}{q}.$$

The same technique—expand (4) for each node in the loop plexus then simultaneously solve for the I values—can be used to calculate metrics for more complicated cases including multiple and/or nested loops. The I values will be passed to the API (Section 3.2.3) to control voltage settings. Later iterations will see less of the budgeted resource remaining so the processor speed will rise or fall, appropriately, as execution continues.

Shin et al [39] require that the maximum iteration counts be known at compile time. When a loop exits before that count is reached, the worst-case expectation for the remaining process complexity is reduced. Zhu et al [46] propose a more sophisticated method of dealing with loops. The loop is unrolled but, depending on inter-iteration dependencies, the individual iterations can be represented as either serial or parallel tasks or some combination thereof. Probabilities are used to specify the likelihoods of actually executing each task cluster. Neither approach deals with the case where the bound on the number of iterations is a priori unknown.

4 Optimizations with General Power Models

The optimizations presented in Section 3 were all developed using the Π_{11} power models. Below, the results of these optimizations using the general Π_{mn} power model described in Section 2.1 are provided. The generalized results are grouped by the process model that is used: first the optimizations for the simple probabilistic model then those for the structural

model. The techniques to derive these more general results are virtually identical to those assuming Π_{11} so they are not repeated.

4.1 Simple Probabilistic Model and Π_{mn}

The simple probabilistic process model posits the existence of a function p , where $p(x)$ is the probability that process complexity is exactly x cycles. The Π_{mn} power model states that $E(x) = \int_0^x v(y)^m dy$ and $T(x) = \int_0^x v(y)^{-n} dy$, where $E(x)$ is the total energy consumed executing cycles $0 \dots x$, $T(x)$ is the total time to execute these cycles, and $v(y)$ is the voltage used to execute cycle y .

The optimization of Section 3.1.1 finds the function, $v(x)$, that achieves the least expected execution time given a not-to-exceed energy budget E using Π_{11} . The results of that optimization are captured in (1). When the Π_{mn} power model is used instead, the corresponding results are

$$v(x) = z(x)^{\frac{1}{m+n}} \left[\frac{E}{\Phi(m)} \right]^{\frac{1}{m}} \quad \text{Ex}(T) = \frac{\Phi(m)^{\frac{m+n}{m}}}{E^{\frac{n}{m}}} \quad \text{Ex}(E) = \frac{\Phi(2m+n)}{\Phi(m)} E, \quad (1')$$

where

$$\Phi(r) = \int_0^c z(x)^{\frac{r}{m+n}} dx.$$

Note that $\Phi(r)$ is a non-increasing function of r .

The minimization of Section 3.1.2 is similar except that there is a not-to-exceed time budget, T , and the objective is to minimize expected energy consumption. The results of the optimization, corresponding to (2), using the Π_{mn} power model are

$$v(x) = \frac{1}{z(x)^{\frac{1}{m+n}}} \left[\frac{\Phi(n)}{T} \right]^{\frac{1}{n}} \quad \text{Ex}(E) = \frac{\Phi(n)^{\frac{m+n}{n}}}{T^{\frac{m}{n}}} \quad \text{Ex}(T) = \frac{\Phi(m+2n)}{\Phi(n)} T, \quad (2')$$

where Φ is as defined above.

The optimization of Section 3.1.4 seeks to minimize the expected value of the penalty function $Q(E, T)$. The condition for a stationary solution, corresponding to (3), using the Π_{mn} power model, is

$$v(x)^{m+n} = \frac{n \int_0^c p(y) Q_2(y) dy}{m \int_0^c p(y) Q_1(y) dy}. \quad (3')$$

Energy delay metrics of the form $Q(E, T) = E^\alpha T^\beta$, often used to measure architecture efficiency [23, 29, 47], are anomalous when used as an application metric. When one checks for

a constant voltage stationary solution using the above formula, v terms cancel and $m\alpha = n\beta$ is the residual. When the equality is true, *any* constant voltage produces the same expected value of Q ; when the equality is false, either the maximum or minimum possible voltage will minimize Q .

4.2 Structural Process Model and Π_{mn}

Section 3.2 developed optimizations given a (σ, S, c_x, p) structural process model using the Π_{11} power model. This section presents the results of those optimizations when an arbitrary Π_{mn} power model is used instead.

Section 3.2.1 finds the optimal voltage to use for each $s \in S$ and the expected execution time when there is a not-to-exceed energy budget. The results analogous to (5) when the Π_{mn} power model is used are

$$v(s) = \left[\frac{e}{I(s)} \right]^{\frac{1}{m}} \quad \text{Ex}(t(s)) = \frac{I(s)^{\frac{m+n}{m}}}{e^{\frac{n}{m}}} \quad (5')$$

where e is the energy remaining when s is executed and

$$I(s) = c_x(s) + \left(\sum_{r \in \theta(s)} p(s, r) I(r)^{\frac{m+n}{m}} \right)^{\frac{m}{m+n}}. \quad (8)$$

Section 3.2.2 finds the optimal voltage to use for each $s \in S$ and the expected energy utilization when there is a not-to-exceed time budget. The results analogous to (7) when the Π_{mn} power is used are

$$v(s) = \left[\frac{I(s)}{t} \right]^{\frac{1}{n}} \quad \text{Ex}(e(s)) = \frac{I(s)^{\frac{m+n}{n}}}{t^{\frac{m}{n}}} \quad (7')$$

where t is the time remaining when s is executed and

$$I(s) = c_x(s) + \left(\sum_{r \in \theta(s)} p(s, r) I(r)^{\frac{m+n}{n}} \right)^{\frac{n}{m+n}}.$$

Note that forms such as $(\sum \alpha_i x_i^r)^{1/r}$ are known as general weighted (or Hölder) means and are increasing functions of r [17].

5 Metric Estimates and Deployment

Implementing the optimizations developed above requires metrics about the hardware platform to determine the proper power model to use and metrics about the application to model

it and compute an optimal voltage schedule. Hardware performance metrics and tradeoffs are typically documented by chip vendors and can be supplemented with simulation and testing.

Information about applications is more specific and must be developed on a case-by-case basis. Pouwelse and Langendoen [35] argues that voltage scaling can only be effective when applications cooperate and Barnett [4] describes how applications form energy-related tradeoffs to measure and enhance system performance. Sources of applications metrics include developer intuition and domain knowledge, simulation, profiling, and feedback from the executing systems. Section 6 discusses the impact of errors in estimating these metrics. The remainder of this section briefly discusses methods to collect application-specific information and embed it in a system.

The most straightforward way to predict the execution time of a task is to gather execution times of previous instances of the same task [24]. For example, [46] assumes that probabilities and complexities not known a priori are determined by profiling. Lorch and Smith [27] discuss and analysis sophisticated methods to gather probabilities for the simple process model. The tradeoffs of using recent versus long term statistics are explored along with how best to model the data collected, i.e., for what sort of distribution—normal, Pareto, etc.—should the parameters be estimated. It is argued that the goal of data collection and deployment is primarily to optimize system performance, not necessarily to capture the best representation of the distribution. The fact that the probabilities may not be stationary and could change over time as well as the difficulties of estimating meta-distributions are also noted. Barnett [5] discusses the value and costs of using meta-distributions at compile and runtime in a general problem-solving context.

Once the application metrics have been gathered and the proper power model selected, there remains the problem of inserting power management points in the application. Designing, developing, and maintaining realtime code or any code that directly interacts with hardware devices are expensive error-prone activities [7]. Thus, approaches that shield the majority of the developers from the details of power and time management are needed to reduce costs and implement more robust power-aware systems. There are many approaches to technology insertion.

Some methods do optimizations and insertion offline [44] or place the entire burden for power management, online, in the operating system [15] or the scheduler [27]. Other

approaches rely on power-aware compilers [25, 32, 39] which certainly seems best when feasible. Many compilers already use hardware models to count cycles and are aware, through dataflow analysis, of when future branch decisions are actually determined. This enables use of information about future process complexity as early as possible. Section 6.2 estimates the value of using such information promptly and the cost if there is a delay. It should also be possible to form power-aware versions of programming languages where pragmas can be used to specify crucial application characteristics such as branch probabilities and sources of energy and time budgets.

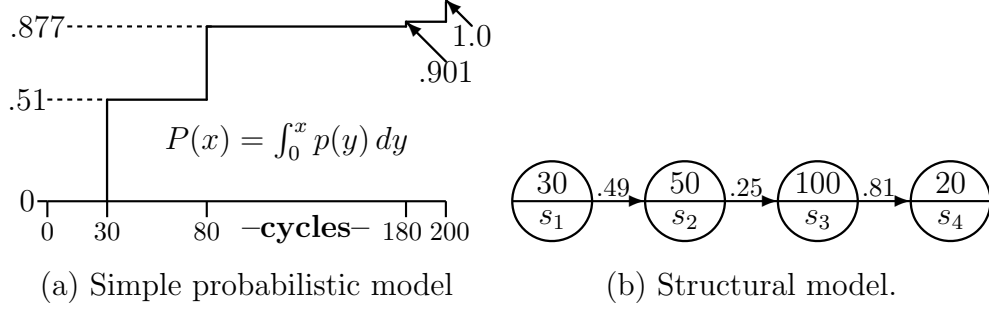
Assuming the power management mechanism is not buried beneath the application, the developers must still decide how often and where power management points appear in code, e.g. as calls on a power management API. (See Section 3.2.3.) The simplest approach is to adjust voltage at the release of each code segment or task [3, 15, 28, 39, 43, 46]. Lorch and Smith [27] discusses methods to break a task into chunks—each initiated by a separate power management point—using the simple probability model function, p , to guide the segmentation. It is noted that breaking tasks into ten segments, each scheduled at a constant voltage, is sufficient to get within 1.2% of an optimal energy solution where agile voltage control is assumed. Whatever strategy is used to place power management points in code, automated or semiautomated tools should be considered to ease the burden on application developers.

6 Analysis of Results

The formulas derived for optimal voltage scheduling are analyzed in this section. Section 6.1 compares the behavior of optimal schedules with schedules derived by considering average-case behavior. Section 6.2 analyzes the benefits of using information about future complexity promptly. Section 6.3 analyzes the impact on optimal performance when model parameters are misestimated. Finally, Section 6.4 develops theoretical criteria for when voltages change in an optimal schedule and when they should remain constant.

6.1 Estimates are Conservative

Figure 5(a) and 5(b) show, respectively, a simple probabilistic model represented by $P(x) = \int_0^x p(y) dy$ and a structural model for the same process. That process consists of the segments



	s_1	s_2	s_3	s_4	max	Ex
$A(s_i)$	68.734	79.050	116.200	20.000		
$v(s_i)$	1.455	0.713	0.178	0.144		
ΔE	43.646	35.644	17.822	2.887	100.000	63.582
ΔT	20.620	70.137	561.098	138.543	790.398	137.469
$I(s_i)$	106.300	109.000	118.000	20.000		
$v(s_i)$	0.941	0.659	0.329	0.296		
ΔE	28.222	32.926	32.926	5.927	100.000	48.977
ΔT	31.890	75.929	303.714	67.492	479.025	112.997

(c) Metrics for average case and optimal strategies with $E = 100$.

Figure 5: Comparison of average case and optimal strategies.

s_1, \dots, s_4 . Each is executed in turn and the process may terminate after any of the segments. Optimal scheduling is compared, using this process example, with the intuitive idea of using average-case estimates of the remaining computation to set voltage levels [30, 46]. For simplicity, the Π_{11} model is assumed.

The average expected complexity of the process starting at s_i is $A(s_i) = c_x(s_i) + p(s_i, s_{i+1})A(s_{i+1})$, where $A(s_4) = c_x(s_4)$. If the problem were to reduce expected execution time within a total energy budget, E , the voltage used at s_i would be $v(s_i) = e_i/A(s_i)$, where e_i is the energy remaining just before s_i is executed and $e_1 = E$. The optimal strategy is $v(s_i) = e_i/I(s_i)$, where $I(s_i) = c_x(s_i) + p(s_i, s_{i+1})^{1/2}I(s_{i+1})$ and $I(s_4) = c_x(s_4)$. (This is just (4) specialized to the example process.) Since $A(s_i) < I(s_i)$ except for the trivial cases where $p = 1$ or $p = 0$, the voltage selected by the average-case strategy at s_i will be greater given the same e_i . However, the energy budget remaining if s_{i+1} is executed will be less. Thus one may be motivated to describe the optimal strategy as conservative in that it reserves more energy for future contingent executions. Note that (1) these observations

follow from (8) so are valid for *all* power models and (2) $A(s_i) < I(s_i)$ remains valid for more complicated structural models too.

Figure 5(c) shows metrics associated with the execution of each s_i as well as the expected total execution time and energy expenditure. The rows labeled ΔE and ΔT report, respectively, the energy used to execute s_i and the time it takes. The maximum energy used by both strategies is $E = 100$ when all segments execute. However, the expected energy expenditure is 23.0% less if the optimal strategy is used. When delays are compared, the optimal strategy reduces the maximum execution time by 39.4% and the expected execution time by 17.8%.

6.2 Promptness is a Virtue

It was noted in Section 2.2.3 that the structural model often provides more information than a simple probabilistic model for the same process. The reason stated was that information about future behavior would be available sooner so that more informed voltage scheduling decisions could be made. An example is used to illustrate the value of prompt information deployment.

Consider a process whose initial segment complexity is $2a$ with a second segment of complexity ra executed with probability p . The expected time to execute this process, assuming the Π_{11} model, is I^2/E , where $I = 2a + \sqrt{p}ra$ and E is the total energy budget. What is the effect if the decision to execute the second segment or terminate were known after a cycles? In this case, a more informed model would posit an initial segment of complexity a with a branch to one of two final segments: one segment, with complexity $a + ra$, would be executed with probability p and the second, with complexity a , would be executed with probability $q = 1 - p$. Here, the expected process execution time is J^2/E , where $J = a + \sqrt{p(a + ra)^2 + qa^2}$. It is easy to show that $J < I$ unless $p = 0$, $p = 1$, or $r = 0$. The question is what fraction of the expected execution time does early information availability and use save? The fraction of time saved is simply

$$\frac{I^2/E - J^2/E}{I^2/E} = 1 - \left(\frac{1 + \sqrt{p(1+r)^2 + q}}{2 + \sqrt{p}r} \right)^2.$$

Figure 6 shows this fraction for several values of p as a function of r . The maximum potential savings increase as p becomes smaller, however, the r where the maximum occurs becomes

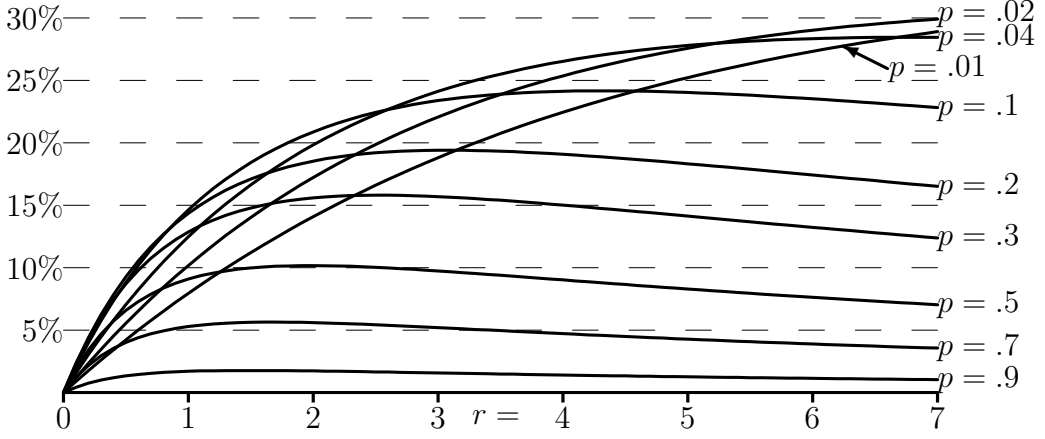


Figure 6: Fraction of execution time saved by prompt information use.

larger. The reason is that extra energy is committed to the second a -cycle segment, but in the rare case where the segment of complexity ra is executed, the penalty is rather large. Similar findings following when the general Π_{mn} model is used and when time rather than energy is budgeted. The minimization process is accelerated when new information is used promptly. While it may be difficult to manually optimize code to use information as soon as possible, it is the sort of task that could be performed by a power-aware compiler through dataflow and branch analysis.

6.3 Robustness of Results

Estimates of various probabilities and execution complexity are based on approximate models and limited testing so might not be exact. (See Section 5.) Further, the Π_{mn} power models ignore threshold voltage, variable states of caches and pipelines, and environmental influences all of which effect the energy vs time tradeoff. This section investigates some of these sources of inaccuracies and their impacts on claims of optimality for the voltage scheduling techniques developed above. While these techniques will be shown to be robust, one must always be cautious when applying theory to practical cases that are not exactly modeled. The Π_{11} model will be used to simplify the analyses but note that the form of the results and qualitative assertions carry over to the general case.

6.3.1 Simple Distribution Estimation Errors

Assume that $p(x)$ is an accurate simple probabilistic model and a fixed energy budget E is given. Then from (1), we know that $\text{Ex}(T) = I^2/E$ and $\text{Ex}(E) = E \int_0^c z(x)^{2/3} dx / I$. The issue to be investigated here is the effect on $\text{Ex}(T)$ and $\text{Ex}(E)$ if some $p_1 \neq p$ were used, instead of p , to determine the voltage schedule. That voltage schedule would be $v_1(x) = Ez_1(x)^{1/2}/I_1$, where $z_1(x) = 1 - \int_0^x p_1(y) dy$ and $I_1 = \int_0^c z_1(x)^{1/2} dx$. Therefore, the expected resource utilizations are

$$\begin{aligned} \text{Ex}(E_1) &= \int_0^c z(x)v_1(x) dx & \text{Ex}(T_1) &= \int_0^c z(x)/v_1(x) dx \\ &= \frac{E}{I_1} \int_0^c z(x)z_1(x)^{1/2} dx & &= \frac{I_1}{E} \int_0^c \frac{z(x)}{z_1(x)^{1/2}} dx \end{aligned}$$

and the measures we seek are

$$\text{Er}(E) = \frac{\text{Ex}(E_1) - \text{Ex}(E)}{\text{Ex}(E)} \quad \text{Er}(T) = \frac{\text{Ex}(T_1) - \text{Ex}(T)}{\text{Ex}(T)},$$

which are the fractions of expected additional energy and time consumed if p_1 is used instead of the true density function p . Of course these measures can be large if p and p_1 are substantially different.

If p and p_1 are reasonably alike, e.g., if it is assumed that (1) $I = I_1$, (2) $|z_1(x)^{1/2} - z(x)^{1/2}| < \delta$, and (3) $z(x)^{1/2}/z_1(x)^{1/2} < 1 + \epsilon$, then bounds on $\text{Er}(E)$ and $\text{Er}(T)$ are straightforward to calculate: namely,

$$\text{Er}(E) < \frac{\delta \text{Ex}(c)}{\int_0^c z(x)^{3/2} dx} \quad \text{Er}(T) < \frac{\delta(1 + \epsilon)c}{I},$$

where $\text{Ex}(c) = \int_0^c z(x) dx$ is the expected process complexity. Both bounds are reasonable small if ϵ and δ are small and p does not have an extremely long, low-valued tail. It is interesting to compare this conclusion with the discussion in Section 6.2 of worst case behavior exhibited in Figure 6.

6.3.2 Structural Model Errors

Executions based on the structural process model are robust in that they optimally recover from estimation errors. That robustness is serendipity from the API algorithm shown in Figure 2. Even if earlier execution has been based on misestimated c_x and p values, the optimal continuation is to execute segment s with voltage $e/I(s)$ or $I(s)/t$, when $I(s)$ is

correctly estimated, because e , respectively t , is the actual remaining resource. If, however, e , respectively t , were projected by offline analysis, the prior errors and divergence from optimality would be exacerbated.

The remainder of this section considers the effect when a set of branch probabilities are misestimated and the objective is to minimize expected execution time given an energy budget. The Π_{11} power model will be assumed for simplicity, but note that similar results are available for the general Π_{mn} model and the problem where time is the constraint.

Consider the situation just before segment s is executed with remaining energy e . The total expected remaining execution time, from (5), is $\text{Ex}(T) = I(s)^2/e$, where $I(s) = c_x(s) + \psi$ and $\psi = \sqrt{\sum_{x \in \theta(s)} p(s, x) I(x)^2}$. Assume that the $p(s, x)$ are misestimated so that $I_1(s) = c_x(s) + \psi_1$ is believed instead of the true value. Thus, s will be executed with voltage $v_1 = e/I_1(s)$ and consume energy $e_1 = ec_x(s)/I_1(s)$ leaving energy $e_2 = e - e_1 = e\psi_1/I_1(s)$ for the remaining execution. Executing s will take time $t_1 = c_x(s)/v_1 = c_x(s)I_1(s)/e$.

Since the $I(x)$, where $x \in \theta(s)$, are assumed correct—only the p values are suspect—the actual expected remaining execution time is $t_2 = \psi^2/e_2 = I_1(s)\psi^2/(e\psi_1)$. Therefore, the actual expected total execution time with the false assumption is

$$\begin{aligned} \text{Ex}(T_1) &= t_1 + t_2 \\ &= \frac{(c_x(s)\psi_1 + \psi^2)(c_x(s) + \psi_1)}{\psi_1 e}. \end{aligned}$$

The relative loss from the false assumption is defined to be

$$\text{Er}(T) = \frac{\text{Ex}(T_1) - \text{Ex}(T)}{\text{Ex}(T)}.$$

Now define α and β such that $\psi = \alpha c_x(s)$ and $\psi_1 = \beta c_x(s)$ and substitute in the above to show that

$$\text{Er}(T) = \frac{(\alpha - \beta)^2}{\beta(1 + \alpha)^2}.$$

The objective is to analyze $\text{Er}(T)$ in terms of the estimation error. Define $r = \alpha/\beta$ and rewrite the above as

$$\text{Er}(T) = \frac{\beta(r - 1)^2}{(1 + r\beta)^2}.$$

If $|r - 1| < \epsilon$, for some $\epsilon > 0$,

$$\text{Er}(T) < \frac{\beta\epsilon^2}{(1 + r\beta)^2} \leq \frac{\epsilon^2}{4r},$$

where the second inequality follows because the maximum value of $\beta/(1 + r\beta)^2$ occurs when $\beta = r^{-1}$. Thus, if ϵ is small so is $\text{Er}(T)$.

6.4 When Can/Should Voltage Change?

A simple and important theoretical result follows immediately from the developments in Section 4.1. Voltage should only change in an optimal voltage schedule when executing in a region where the probability of termination is nonzero. Table 3 summarizes the voltage

Table 3: Summary of agile voltage scheduling results.

min	$\text{Ex}(T)$	$\text{Ex}(E)$	$\text{Ex}(Q(E, T))$
$v(x) =$	$z(x)^{\frac{1}{m+n}} \left[\frac{E}{\Phi(m)} \right]^{\frac{1}{m}}$	$\frac{1}{z(x)^{\frac{1}{m+n}}} \left[\frac{\Phi(n)}{T} \right]^{\frac{1}{n}}$	$\left[\frac{n \int_0^c p(y) Q_2(y) dy}{m \int_0^c p(y) Q_1(y) dy} \right]^{\frac{1}{m+n}}$

computations for simple probabilistic models where agile voltages—those that can change anywhere—are considered. Assume that $p(x) = 0$ when $x_1 \leq x \leq x_2$. Then $z(a) = z(b)$ for any $x_1 \leq a, b \leq x_2$ because $z(x) = 1 - \int_0^x p(y) dy$. Therefore, the result is immediate when the objective is to minimize $\text{Ex}(E)$ or $\text{Ex}(T)$. It is also immediate when the objective is to minimize $\text{Ex}(Q)$ because $\int_a^c p(y) Q_i dy = \int_b^c p(y) Q_i dy$ when p , a , and b have the assumed properties. Another way to summarize this fact is that voltage will remain constant unless something new is learned; what can be learned during the execution of cycle x is that the process did or did not terminate at cycle x . Such a discrimination is only possible when $p(x) \neq 0$.

Similar observations follow when optimal voltage schedules are derived using a structural process model. Voltage never changes during the execution of a segment. However, when new information becomes available about process complexity by choosing a branch, the voltage level might change. So the general conclusion, using either model, is that voltage can only change in an optimal schedule when something new is learned about the future complexity of the process.

References

- [1] *Advanced Micro Devices Corporation*, <http://www.amd.com>, March 2004.
- [2] *AeroVironment Corporation*, <http://www.aerovironment.com>, March 2004.
- [3] H. Aydin, R. Melhem, D. Mossé, P. Mejía-Alvarez, Determining optimal processor speeds for periodic real-time tasks with different power characteristics, *Euromicro Conference on Real-Time Systems*, pp. 225–232, 2001.
- [4] J. Barnett, Application-level power awareness, In R. Graybill and R. Melhem (Eds.), *Power Aware Computing*, pp. 227–242, Kluwer, 2002.
- [5] J. Barnett, How much is control knowledge worth? A primitive example, *Artificial Intelligence*, 22(1), pp. 77–89, 1984.
- [6] K. Błażewicz, E. Ecker, E. Pesch, G. Schmidt, and J. Węglarz, *Scheduling Computer and Manufacturing Processes*, Springer-Verlag, Berlin, Germany, pp. 346–350, 1996.
- [7] B. Bohem, E. Horowitz, R. Madachy, D. Reifer, B. Clark, B. Steece, A. Brown, S. Chulain, C. Abts, *Software Cost Estimates with COCOMO II*, Prentice Hall, 2000.
- [8] R. Buck, *Advanced Calculus*, McGraw Hill, 1956.
- [9] T. Burd and R. Brodersen, Energy efficient CMOS microprocessor design, *HICSS Conference*, pp. 288–297, 1995.
- [10] T. Burd, T. Pering, A. Statakis, and R. Brodersen, A dynamic voltage scaled microprocessor systems, *IEEE Journal of Solid-State Circuits*, 35(11), pp. 1571–1580, 2000.
- [11] G. Cao, Proactive power-aware cache management for mobile computing, *IEEE Transactions on Computers*, 51(6), pp. 608–621, 2002.
- [12] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, Data driven signal processing: An approach for energy efficient computing, *ISLPED*, 1996.
- [13] A. Chandrakasan, S. Sheng, and R. Brodersen, Low-power CMOS digital design, *IEEE Journal of Solid-State Circuits*, 27(4), pp. 473–484, 1992.

- [14] D. Gillies and W. Liu, Scheduling tasks with AND/OR precedence constraints, *SIAM Journal of Computing*, 24(4), pp. 797–810, 1995.
- [15] F. Gruian, Hard real-time scheduling for low-energy using stochastic data and DVS processors, *ISLPED*, 2001.
- [16] V. Gutnik and A. Chandrakasan, An efficient controller for variable supply voltage low power processing, *Symposium on VLSI Circuits* pp. 158–159, 1996.
- [17] G. Hardy, J. Littlewood, and G. Pólya, *Inequalities*, Cambridge University Press, 1988.
- [18] M. Igarashi, K. Usami, K. Nogami, F. Minami, Y. Kawasaki, T. Aoki, M. Takano, C. Mizuno, T. Ishikawa, M. Kanazawa, S. Sonoda, M. Ichida, and N. Hatanaka, A low-power design method using multiple supply voltages, *ISLPED*, 1997.
- [19] *Intel Corporation*, <http://developer.intel.com/design/intelxscale>, March 2004.
- [20] Intel, Microsoft, and Toshiba, *Advanced Configuration and Power Management Interface (ACPI) Specification*, <http://www.acpi.info>, March 2004.
- [21] *International Business Machines*, <http://www.ibm.com>, March 2004.
- [22] T. Isihara and H. Yasuura, Voltage scheduling problems for dynamically variable voltage processors, *ISLPED*, pp. 197–202, 1998.
- [23] J. Jalminger and P. Stenström, Improvement of energy-efficiency in off-chip caches by selective prefetching, *Microprocessors and Microsystems* 26(3), pp. 107–121, 2002.
- [24] P. Kumar and M. Srivastava, Predictive strategies for low-power RTOS scheduling, *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Austin, TX, 2000.
- [25] S. Lee, and T. Sakurai, Run-time voltage hopping for low-power real-time systems, *37th Design Automation Conference*, pp. 806–809, 2000.
- [26] D. Li, P. Chou, and N. Bagherzadeh, Topology selection for energy minimization in embedded networks, *Asia South-Pacific Design Automation Conference (ASPDAC)*, pp. 693–696, 2003.

- [27] J. Lorch and A. Smith, Improving dynamic voltage scaling algorithms with PACE, *ACM Sigmetrics*, 2001.
- [28] A. Manzak and C. Chakrabarti, Variable voltage task scheduling algorithms for minimizing energy, *ISLPED*, pp. 279–282, 2001.
- [29] A. Martin, M. Nyström, and P. Pénczes, Et^2 : A metric for time and energy efficiency of computation, In R. Graybill and R. Melhem (Eds.), *Power Aware Computing*, pp. 293–315, Kluwer, 2002.
- [30] R. Melhem, N. AbouGhazaleh, and D. Mossé, Power management points in power-aware real-time systems, In R. Graybill and R. Melhem (Eds.), *Power Aware Computing*, pp. 127–152, Kluwer, 2002.
- [31] S. Mohanty, J. Ou, and V. Prasanna, An estimation and simulation framework for energy efficient design using platform FPGAs, *IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2003.
- [32] D. Mossé, H. Aydin, B. Childers, and R. Melham, Compiler-assisted dynamic power-aware scheduling for realtime applications, *Workshop on Compilers and Operating Systems for Low Power*, October 2000.
- [33] W. Namgoang, M. Yu, and T. Meg, A high efficiency variable-voltage CMOS dynamic DC-DC switching regulator, *IEEE International Solid-State Circuits Conference*, pp. 380–391, 1997.
- [34] P. Pillai and K. Shin, Real-time dynamic voltage scheduling for low-power embedded operating systems, *ACM Symposium on Operating Systems Principles*, 2001.
- [35] J. Pouwelse, K. Langendoen, H. Sips, Energy priority scheduling for variable voltage processors, *ISLPED*, pp. 28–33, 2001.
- [36] Q. Qiu and M. Pedram, Dynamic power management based on continuous-time markov decision processes, *Design Automation Conference 36*, pp. 555–561, 1999.
- [37] V. Raghunathan, P. Spanos, and M. Srivastava, Adaptive power-fidelity in energy aware wireless embedded systems, *IEEE Real-Time Systems Symposium*, 2000.

- [38] T. Sakurai and A. Newton, Alpha-power law MOS-FET models and its applications to CMOS inverter delay and other formulas, *IEEE Journal of Solid-State Circuits*, 25(2), pp. 584–594, 1990.
- [39] D. Shin, J. Kim, and S. Lee, Intra-task voltage scheduling for low-energy hard real-time applications, *IEEE Design and Test of Computers*, 18(2), pp. 20–30, March-April 2001.
- [40] P. Shriver, M. Gokhale, S. Briles, D. Kang, M. Cai, K. McCabe, S. Crago, and J. Suh, A power-aware, satellite-based parallel signal processing scheme, In R. Graybill and R. Melhem (Eds.), *Power Aware Computing*, pp. 243–259, Kluwer, 2002.
- [41] K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, and T. Kuroda, 300MIPS/W RISC core processor with variable voltage supply-voltage scheme in variable threshold-voltage CMOS, *Proceedings of the ICC*, pp. 587–590, 1997.
- [42] *Transmeta Corporation*, <http://www.transmeta.com>, March 2004.
- [43] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, Energy-aware runtime scheduling for embedded-multiprocessor SOCs, *IEEE Design & Test of Computers*, 18(5), pp. 46–58, 2001.
- [44] F. Yao, A. Demers, and S. Shenker, A scheduling model for reduced CPU energy, *IEEE Annual Symposium on Foundations of Computer Science*, pp. 374–382, 1995.
- [45] W. Ye, J. Heidemann, and D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, *IEEE Computer and Communications Societies (INFOCOM)*, 2002.
- [46] D. Zhu, D. Mossé, and R. Melhem, Power aware scheduling for AND/OR graphs in real-time systems, *IEEE Transactions on Parallel and Distributed Systems*, (to appear).
- [47] V. Zyuban, P. Kogge, Inherently lower-power high-performance superscalar architectures, *IEEE Transactions on Computers*, 50(3), pp. 268–285, 2001.



AVIONICS FOR THE 21ST CENTURY

Jeff Barnett
jbarnett@nrtc.northrop.com
Northrop Grumman Corporation
Integrated Systems

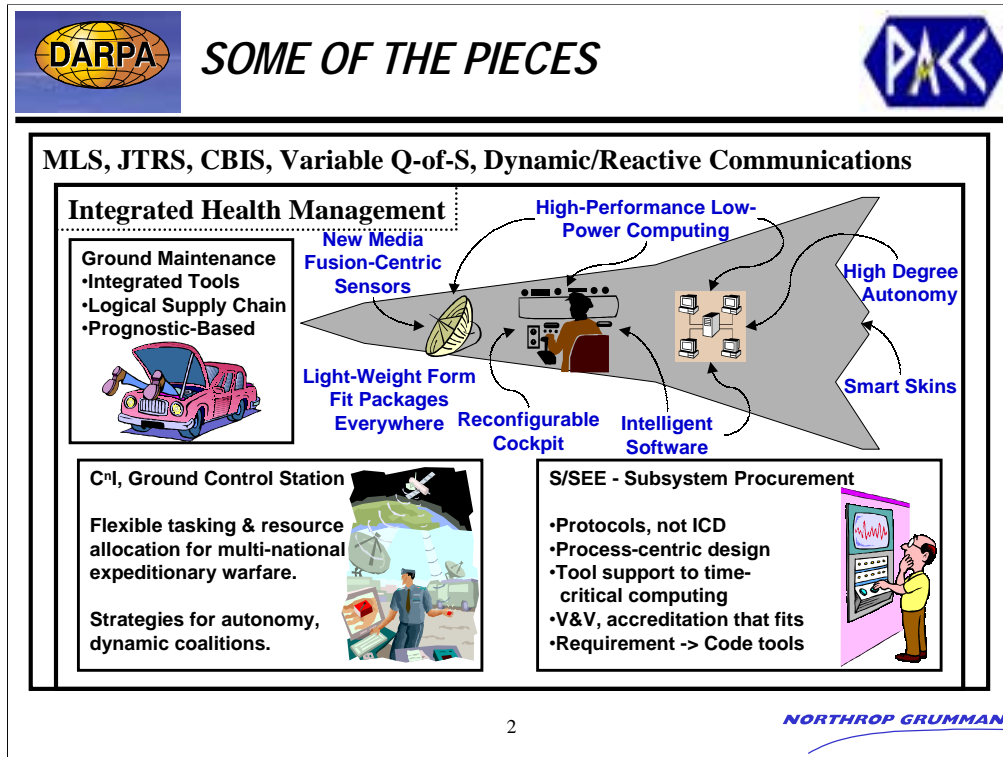
1

NORTHROP GRUMMAN

The information in this briefing was gathered by observing the development of designs for future concept aircraft, consulting the open literature, and reviewing the directions supported by major sources of research funding. The briefing is composed of four parts:

- The first set of slides are about how new technology changes possibilities and requirements and thus presents new opportunities to do things better.
- The second set of slides discusses technology packages that leverage the new potential: computing and local communications architecture, more autonomy and pilot support, improved procurement practices for software subsystems, sensor packaging, reconfigurable cockpits, and smart skins are examined.
- The third set of slides presents engineering challenges, a list of what must be done in particular areas to mature technology, developed in research laboratories or available as COTS, for military applications.
- The fourth set of slides discusses flexible software architectures needed to leverage new hardware capabilities and power. Process (not processor) centric concepts are the theme as are end-to-end communications schemes.

While many of the concepts introduced herein are specific to manned aircraft, most will equally well support unmanned aircraft and other smart weapons systems.



Even though this presentation is titled “Avionics for the 21st Century,” it will address technology that appears in command and control enclaves, ground control stations, maintenance depots, and software engineering environments as well as onboard aircraft. The two outer contours show that revolutions in communications and integrated health management will support all of these domains. The individual labels identify some of the technology packages and approaches discussed herein. In that sense, this is a summary of what follows. The presentation and discussion is about technology, what will be available in the next decade, and how it might be packaged to advantage. A significant omission is a discussion of mission enhancement potential. Of course that is the purpose and goal of supporting a new technology base. However, I believe that the mission benefits will be obvious because the systems will work better, be more robust, and enable new degrees of autonomous behaviors. I also believe that the resulting systems will cost less over their lifecycles. Some of the technology discussed herein will support other domains too. For example, JTRS (software radio) will support ground-mobile troops as well as aircraft. The form factor, available power, and total capabilities will vary but the advantages of flexibility and interoperability will be there. Similarly, low-power, high-performance computing will serve many areas and enable new capabilities and missions.



NEW GROUND RULES FOR 2010



- *Technology changes everything (eventually)*
- *Dumb avionics is no longer an option*
- *Network-centric multinational expeditionary warfare*
- *Sea change in standards (n.b. communications)*
- *DoD wants COTS*
- *Avionics is 30-60% of lifecycle costs*

We have an opportunity to reconsider what avionics is as well as the model of its interaction with the pilot, command staff, and other battlefield components.

3

NORTHROP GRUMMAN

We are all aware of the profound ways that technology advancements have changed civilian, commercial, and government expectations, practices, and procedures. The desire of the military for systems that perform smarter, cost less, and are easier to use is well founded. It is estimated that avionics and software will represent 30-60% of lifecycle costs and it is clear that these areas need attention. The future will see the development of network-centric warfare supported by new standards for the architecture of military systems and the possibility to incorporate COTS products to control costs, simplify technology refresh, and avoid contractor lock-in.

This talk is about the opportunity represented by all of this new technology and how we might be able to reap its promises. The goals are to provide better ways to support pilots, virtual pilots, command staffs, and friendly warfighting assets while still controlling costs.



WHAT'S NEW FOR 2010 AVIONICS?



- ***Communications***
 - *JTRS (software radio) for all comm, GPS, and IFFN with reconfigurable antennas*
 - *Global information grid including profound reachback*
 - *Thz+ LAN, fiber or copper — Thz+ WAN*
- ***New media sensors and countermeasures***
- ***Smart skins — embedded sensors, shapers, nets***
- ***Human interface technologies***
- ***Smart software — resilient, flexible organizations***
- ***Low power, high performance computer cores***

4

NORTHROP GRUMMAN

Some of the technologies relevant for avionics of new aircraft are listed on this slide. Software radios and reconfigurable antennas will provide the flexible communications structures needed for network-centric warfare. New sensor organizations will also leverage available high-powered computation devices to provide more flexibility, e.g., hyper-spectral analysis, and better awareness through filtering and fusion as well as newer media such as lasers. Micro-actuators and sensors will allow future aircraft skins to be shaped to control air flow, act as antennas, and cooperate intelligently with health management. Human interface technology, common in homes, offices, and laboratories, will find its way onto aircraft to better support pilots and into ground stations to better support control functions. New and better software technology will support development of new more resilient, more flexible avionics suites that provide more autonomous capabilities. Finally, high performance computing at very low energy costs will dramatically increase computer packing densities. That, in turn, will make it possible to put the requisite capabilities onboard future aircraft. This is necessary enabling technology for new modes of manned flight and it is even more important for autonomous vehicles where computation is the only local substitute for intelligence.



SMART SYSTEMS



- *The right information at the right time used correctly for the right purpose*
- *Flexible, robust organizations that are resilient to component failure, battle damage, and errors*
- *Systems trusted by owners — natural autonomy*
- *Missions understood and executed in context of the entire battlefield*
- *Same infrastructure supports missions other than war*

5

NORTHROP GRUMMAN

In short, the new technology possibilities will make future aircraft smarter. New communications mechanisms and enhanced bandwidth will get the right information to the right place in time to benefit from it. Systems will be more resilient, more capable, and do the right thing more often. Shear increases in computation power will provide new ways to implement redundancy, enhance safety, and perform missions better. Smarter systems will be trusted by their owners to do the right thing at the right time. With that trust, pilot workload will decrease. The goal is “natural autonomy,” functions that are performed by the automation while the human feels in full control. Thus, the goal here is interleaved problem-solving paradigms that produce the correct responses without challenging the authority of the pilot or vehicle control station. When these mechanisms are in place, the information and tools to fully integrate individual systems into the full context of the battlefield will be in place too.



SOME NAMES & NUMBERS TODAY



- ***Computer core: 3.5 watts per Ghz, Power PC***
- ***Memory: 3GB L3 cache for servers***
- ***Memory: PC100 non-volatile; cold to FTB 10 μ -sec***
- ***Displays: paste-on active coatings***
- ***Displays: 2-D & 3-D without goggles***
- ***LAN: Ghz copper home network***
- ***LAN: no loss, preemptable channels (IEEE)***
- ***WAN: Thz military & commercial backbones***
- ***Device: pinhead-sized MEM technology — sensors & actuators — in the substrate or skin***

6

NORTHROP GRUMMAN

The data on this slide is a sample of the technology capabilities available in the COTS world today. The numbers and claims speak for themselves. It is interesting to note that most of this information is available from sources such as PC Magazine. It also must be noted that a technology is not necessarily ready for aircraft deployment just because it is COTS. Much work remains to be done before it can be incorporated on a military aircraft. The point is simply that the suggestions in the rest of this briefing are not pending scientific breakthroughs though significant engineering surely remains to be done.



AN EMBARRASSMENT OF RICHES



- *The technology pipeline is full — so full that choices must be made*
- *Avionics must still be safe, survivable, reliable, and predictable*

We have a logical design that attempts to make those choices while being conservative. It is currently being annotated with metrics. What we have done (are doing) will change as the future comes into sharper focus. (Status)

Systems integrators, such as NGC, must understand these choices in order to provide effective leadership. (Mantra)

7

NORTHROP GRUMMAN

The technology is there to do things different and do things better. Even with the normal 10-15 year lag of military technology behind the commercial world, a revolution in avionics is overdo. In the following slides, coherent ways that the various technologies could be packaged to support that revolution are described. There are many approaches that might be viable but it is important to have a strawman to act as a baseline and that is what is provided herein.

System integrators such as Northrop Grumman must understand these new possibilities in order to provide effective leadership for new system development programs. As we will see later, the new technologies even imply differences in the way that subsystems are procured. In particular, the design and contracting of Operational Flight Programs (OFP) will likely need to change in order to realize the potential of more flexible avionics with less costly technology refresh episodes.



NEW CONCEPTS PACKAGING



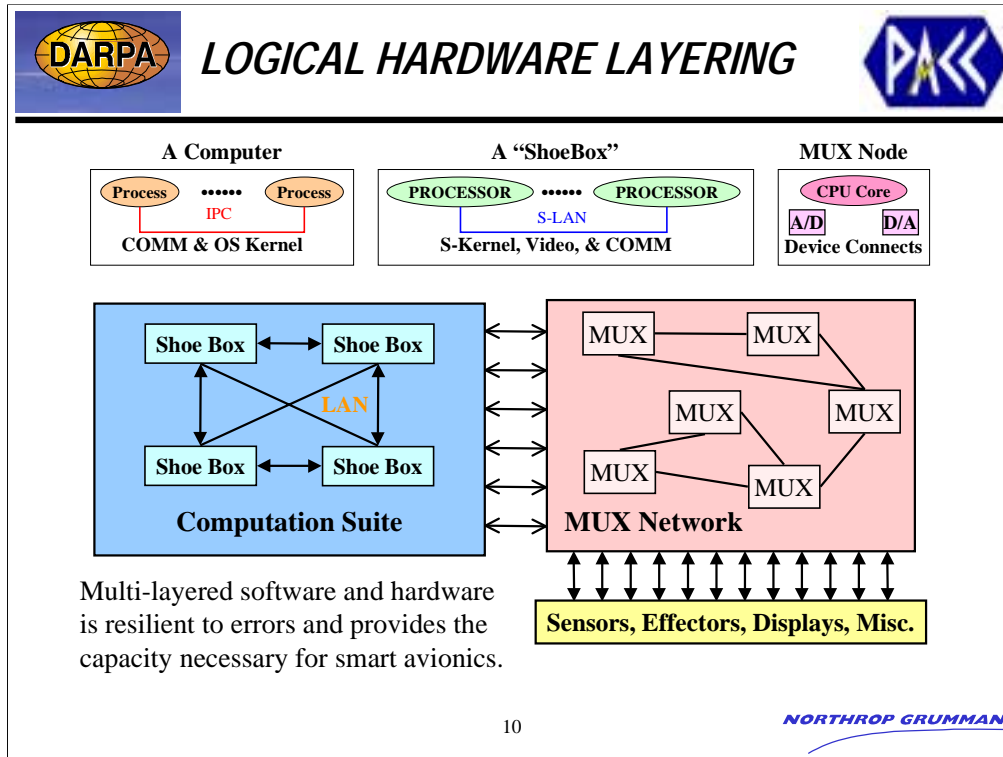
The next group of slides attempt to picture what future avionics will look like. The intent is to group technology in a way that helps imagine the eventual products, the way they will be deployed, and how they are viewed by their users.

The next set of slides portray technology packaged into useful-sized capability chunks. Though specific new technologies can be fitted or retrofitted, one at a time, we will have a better idea of what the future might look like if we consider integrated improvements that can have large impacts on system capabilities. The particular packages examined are: organization of onboard computation and communications facilities, provision of autonomy, OFP architecture and procurement practices, sensor packaging, reconfigurable cockpits, and smart skins. Not all of these concepts will be matured by 2010, e.g., smart skins need more science before they become common. However, the other capabilities only await bold requirements and the desire to see them in the air. Base technologies are already part and parcel of the everyday world outside of the military.

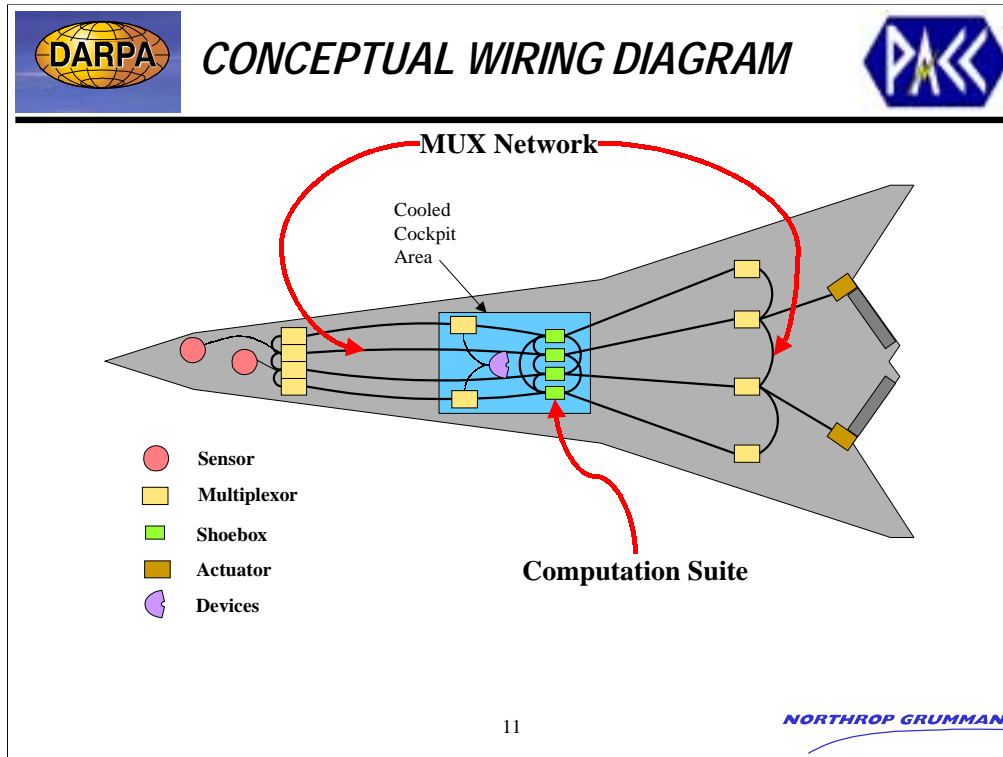


- *Program — The software as written; a lexical entity*
- *Process — An instance of an executing program*
- *Processor — An entity that hosts process executions*
- *Protocol — Rules that organize inter-process interactions & use of communication infrastructure*
- *OFP — A collection of programs, processes, and processors separately contracted; ICD is interface*

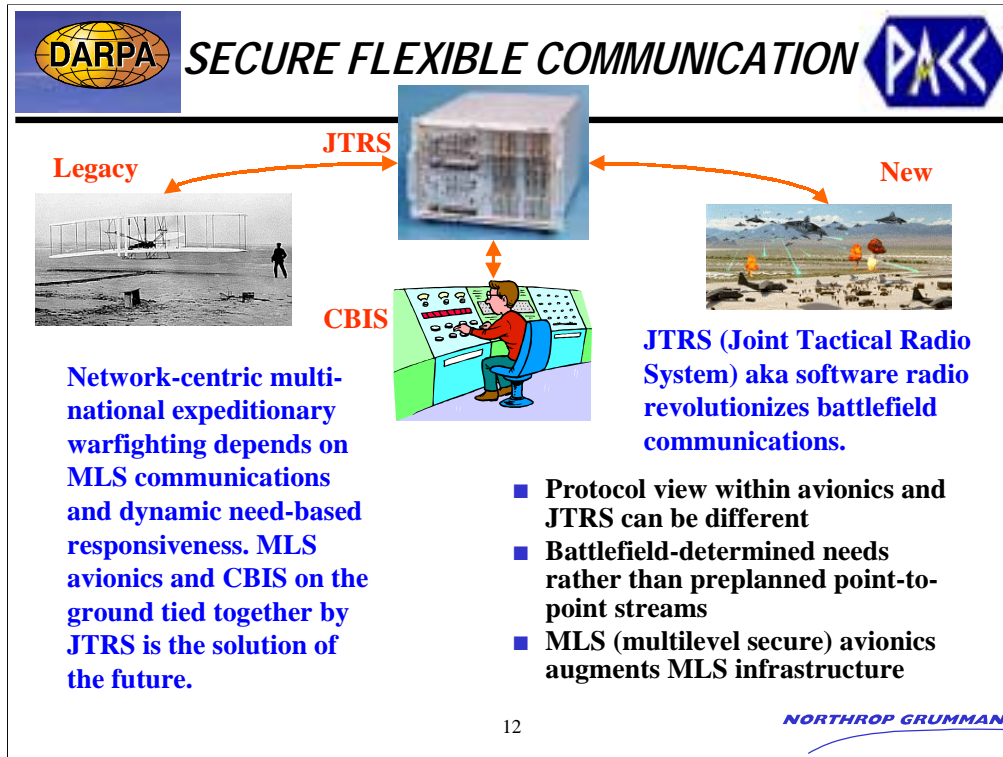
This slide introduces and briefly defines a few terms that describe computation elements. These terms are used throughout the remainder of this briefing. In common parlance, some of them are used interchangeably and that leads to confusion. Different terms are used to describe a program as written and as an executing entity. A process is the name of the latter concept. The term, OFP - Operational Flight Program - is more a contractual term than it is a technical term. Its closest technical equivalent is "software subsystem". However, traditional avionics usually makes the OFP the fundamental architectural unit and this cause problems, particularly with technology refresh.




The hardware layering strategy is portrayed. A set of multiplexors or MUX boxes are located throughout the aircraft and all sensors, effectors, displays, and miscellaneous devices are connected to at least one of them. Each MUX is connected to several others to form a robust data transport mechanism with the main computation suite which consists of several "shoeboxes" interconnected by high-speed fiber. The suite is cooled but the MUX network will operate in uncontrolled areas. Each shoebox contains 5-20 high-speed processors and significant amounts of memory. (Think of a commercial server with multiple CPU chips on a motherboard.) Shoeboxes provide high-speed buses for local communications and software to control the ensemble. Each processor hosts multiple processes. To effect error recovery or balance loads, processes can be restarted on the same computer, moved to another computer in the same box, or even relocated to another box. The set of networks - MUX network, suite LAN, and shoebox LAN - will be capable of dynamic routing which will also add to the robustness of the total solution. In our current design, we expect to have available several times more computation and communication bandwidth than are strictly needed. Those extra resources are traded for reliability and flexibility in system organization. Since hardware capability (via Moore's law) is growing exponentially while avionics requirements are not, our approach should scale into the future, particularly if OTS components can be used.




This slide is a simple plan form cartoon. It shows the shoebox components located in the cockpit area where environment control such as cooling is readily available. The MUX network is scattered throughout the aircraft and has a presence in the cockpit area too. The layouts of the shoebox components will need to respect robustness-against-physical-damage restrictions and the MUX components locations will be determined by available space and a desire to put them close to what they connect.

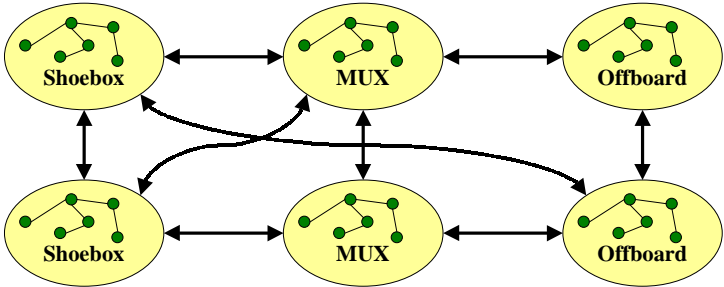


Network-centric expeditionary warfare as envisioned for the future isn't possible without a revolution in the way we communicate. The impediments are of two sorts: technical mechanisms and security issues. This and the next slide discuss these issues. The software radio initiative (JTRS) provides enabling technology for more flexible use of available bandwidth including dynamic allocation and routing. New solutions will be available for legacy as well as those systems built with JTRS in mind. The CBIS (Content-Based Information Security) project at Mitre et al will provide flexible secure infrastructure for console-based systems in command centers and other places. It is also possible with current technology to create multilevel secure avionics suites to complete the picture and we are working hard to make this a reality. Current approaches plan message traffic, allocate communications resources, and establish mission timelines preflight. In the future it will be necessary to support dynamically formed coalitions with the necessary resources at battle time. Since sensor systems as well as shooters will operate opportunistically, there will need to be mechanisms that can determine who needs what data when it is available; the idea of preplanned message traffic would defeat the whole concept. However, none of this will be a reality unless there are flexible dynamic mechanisms in place to provide the requisite security guarantees and flexible communications that provide best use of the available bandwidth.




AN INTERNET ARCHITECTURE





- *An internet is an aggregation of many networks*
- *Logically, that describes our avionics architecture*
 - *Each shoebox is a network*
 - *Each MUX and the devices it connects is a network*
 - *And the offboard assets are other networks*

13



An internet is an aggregation of other networks. That is a fair description of our avionics architecture. Each shoebox is a network comprised of its processors and the processes that they host. Each MUX and the devices that it connects is also a logical network. The fact that some components are on more than one net does not break the metaphor - such components are said to be “multi-homed.” The offboard assets, including other warfighting systems and reachback sites, are also organized as networks that are connected to ownship. This conceptual structure offers many advantages to future avionics designers, implementers, and users: It provides a uniform access paradigm to all resources whatever their physical location, enables the use of OTS technology, and exactly fits the model of the Global Information Grid of Joint Vision 2010 and 2020. To fully enjoy the potential of an internet architecture, many security issues must be addressed. These issues are discussed later in the briefing. It also should be noted that the network interconnections are not and need not be uniform. The shoebox net will probably be a bus structure on a motherboard and the computation suite LAN will be fiber. The MUX net and MUX connections to devices will likely use various sorts of copper cabling at various bandwidths. Connections to offboard assets will use SATCOM, Link 16, etc., either directly or to tunnel connections between onboard and offboard sites. Any future design in this area must factor the huge impact that can be expected from the emergence of software radios (JTRS).



NATURAL AUTONOMY



Fly-By-Wire
Controls




Fly-By-Wire is an example of a natural autonomy technology. The software makes most of the decisions but the user is not aware of the intrusion. The right things just seem to happen. Autonomy in the future will come in packages.

- *The future will see more examples of natural autonomy to support the pilot*
- *Mission planning, targeting, avoidance tactics, fuel management, stealth management, communication with other warfighting entities are candidates*


14

NORTHROP GRUMMAN

Smart weapons systems are desired and needed. However, it is believed by many informed individuals that the most exacting missions (not necessarily the most dangerous) need the judgement of a pilot and, thus, autonomous vehicles are not always appropriate. The question is how to use advanced technology to support the pilot. One of the best examples, fly-by-wire, has been around for a long time. The pilot flies the airplane, decides where it will go, and what it will do. However, automatic mechanisms make decisions about the same things. In fact the automation makes hundreds or thousands of decisions for every one made by the pilot. It is interesting to note that many younger military pilots have never flown a hard-controlled aircraft and are not particularly aware that things might be done differently. The solution not only works, its necessary to achieve the full joint capability of man and machine through symbiosis. We call this approach “natural autonomy.” It is clear that natural fusion of capabilities are needed in many other areas. Modern warfare is predicated on communications infrastructure and complicated plans being executed by separate assets. There is too much to know and far too much information to incorporate for an unaided pilot to make use of his system to best advantage. Natural autonomy is the way to properly support the pilot with automation.

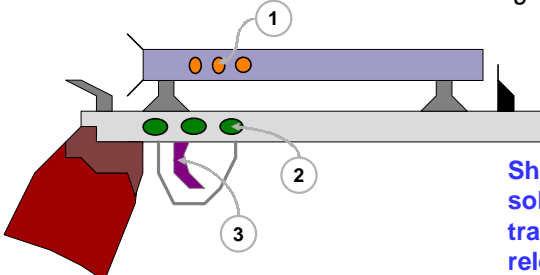


**NATURAL AUTONOMY PACKAGE:
UCAR**



- 1 Shooter selects and aims at target**
UCAR team gathers & transmits imagery
- 2 Shooter selects munition**
UCAR team readies stores
- 3 Shooter shoots**
UCAR team launches munition and guides to target

Natural autonomy is technology package where user need not be constantly aware of autonomous aspects of system.



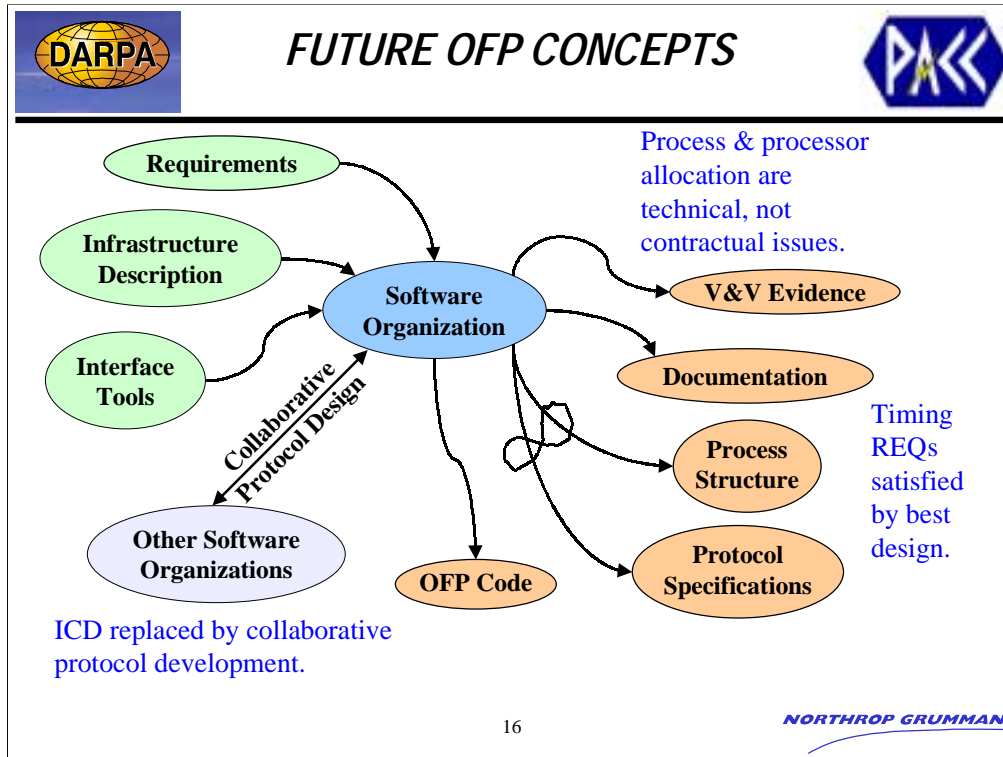
All-electronic rifle packaging achieves natural interface

Shooter - Apache backseater or ground soldier - uses a known interface. Less training, nearly appropriate weapons release doctrine in place, simplifies all operational aspects.

NORTHROP GRUMMAN

15

This slide portrays an extreme example of the natural autonomy concept. A single soldier uses an all-electronic rifle to control the activities of several UCARs (Unmanned Combat Armed Rotorcraft). The image in the rifle scope is formed in realtime from sensors onboard UCARs. As the scope moves, so must the UCARs to maintain the correct imagery. When a munition is selected by a button on the rifle (step 2), UCARs with that weapon must move to firing position and ready that weapon for launch. In step 3 the trigger is pulled and the UCARs release the weapon. In steps 2 and 3, sensor UCARs must initiate target tracking and in step 3 guidance must be provided. At the end of step 3, the sensor UCARs must provide battle-damage assessment images to the soldier through the rifle scope. There are many under-the-cover autonomous actions that the UCAR fleet must perform: allocating vehicles to the various tasks for one or more rifle holders, checking doctrine as well as a clear-line-to-target before firing, coordinating flight, performing vehicle health management diagnostics, and much more. The point is that packaging such as suggested by this example greatly simplifies the soldier's job; he concentrates on war fighting while the technology looks after itself. It should be noted that technology refresh is also simplified. In most cases, the users are not involved and when they are it is to learn about a new capability. I make no claim whether or not this technology will be available in the next decade or so. I present it as an example of where technology, properly integrated with its human users, can go.



Today's avionics systems are comprised of separate subsystems called OFPs. Requirements are allocated and interfaces designed very early in the lifecycle. Usually legally separable organizations - different companies or divisions in the same company - are contracted to build an OFP to spec. In many instances OFPs are allocated distinct processors. This arrangement is a root causes of extreme development costs and resilience of avionics against technology refresh. Note however that this is a logical outcome of not enough computer resources and inadequate buses such as the 1553A. We can do better in the future.

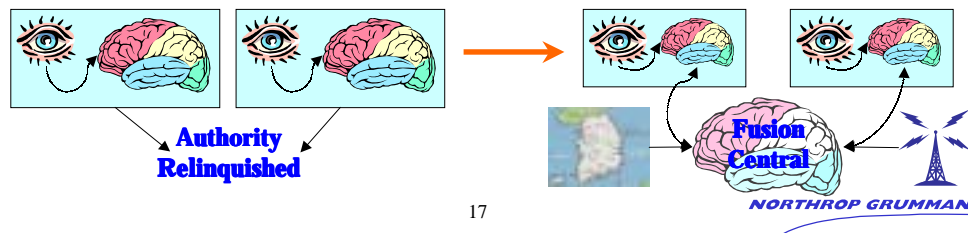
The modern approach is a process-, not processor-centric architecture with the use of end-to-end protocols to glue the system together. The interface will be the product of best technical design rather than an early response to lack of adequate resources. The organizations involved will co-design their system interfaces and design decisions and dependencies will be exposed and confined to those who actual have a stake. In the current mode, it is almost impossible to predict how a change in one part of a system will effect others. In the model proposed here, the shared part of a design is a process architecture and protocol specification. This also permits the runtime system to decide where to best execute processes, how to provide adequate communications, and how to implement general reliability and error recovery methods including the deployment of redundancy.



SENSOR PACKAGING

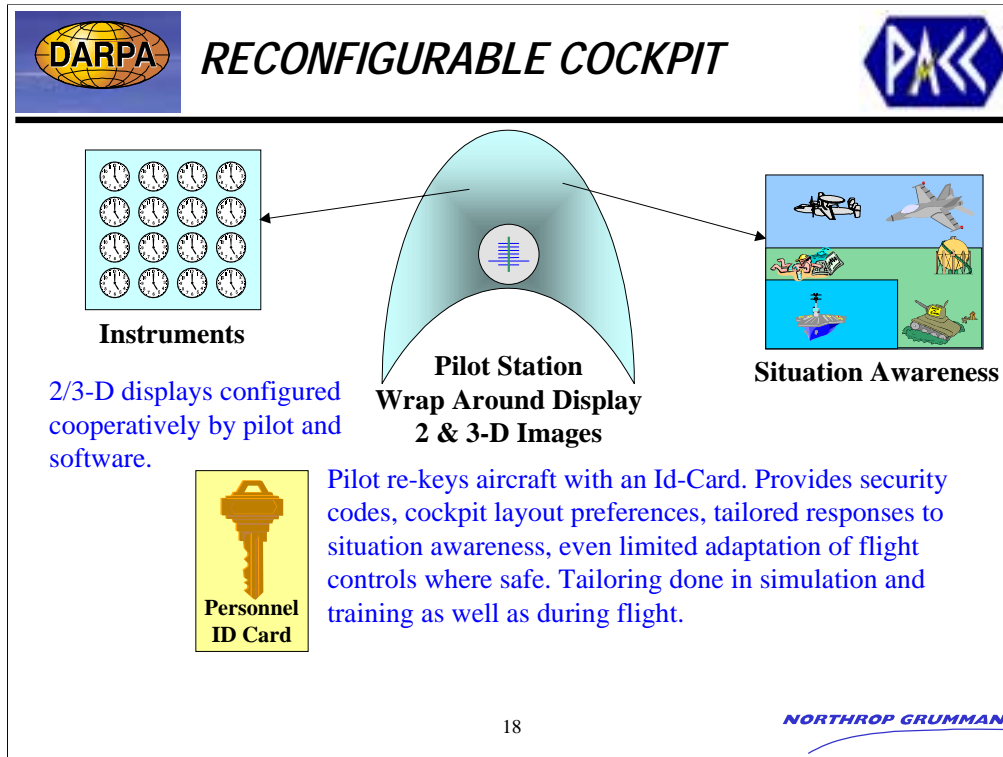


- *More types: IR, radar, laser, EM, visual, hyper-spectral, acoustics*
- *More onboard processing & fusion: share with networked assets for coherent ownship situation*
- *Software sensors by analogy to software radio:*
 - *Current - sharable multiplexed radars, hyperspectral IR*
 - *Less in the black boxes: more use of common processors*
 - *Interest queuing & fusion at deeper representation levels*




17

One of the best ways to spend the large increment in computation power is by doing better sensor processing. Network-centric approaches will provide many more information sources to be fused with data generated on board. Extra power can be used to fuse sensor data at a lower level, e.g., data now turned in to tracks within a sensor can be directly fused with data from other sources. It will also be possible to use richer map data and fuse sensor information provided by other dynamic information-gathering assets. Further, by analogy to software radios, we can construct software sensors. (The computers in sensors will be more powerful too.) The software dynamically loaded in a sensor morphs it to the best configuration for the current situation. Maintenance and technology refresh is simplified if more of the filtering and fusion activities can be moved to the central computer suite (the shoeboxes) where adequate computer power in addition to more information is available. The inherent risk is a potential lack of incentive and cooperation from the niche contractors who make sensors. The reward is a revolution in the information product produced for situation awareness, sensor cueing, and response formation.




Another place to invest enhanced resources is in a reconfigurable cockpit. LED coatings allow arbitrary surfaces to be displays, so large contiguous areas are possible. The display will show instruments and gauges in addition to situation awareness and other common formats. Pilots can arrange the displays to their tastes and request specific configurations in response to specific events. It is even possible to tune the “feel of flight” to individual pilot preferences via the use of customized control laws. In our model, pilots carry electronically-readable personnel ID cards that contain their accumulated preferences. A single aircraft can present itself differently to different crews and different planes will behave the same for the same crew. Advances in display technology will permit limited 3D images without goggles. Gaze tracking apparatus are probably going to be part of future human interfaces and will provide unprecedented amounts of digestible information to the pilot. The technology to implement this vision of a reconfigurable cockpit is at hand. The next step is the human factors engineering necessary to make it a reality.

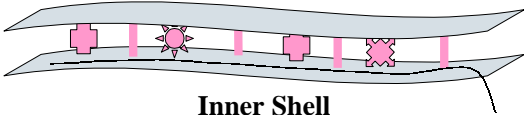
On a technical note, we expect graphics cards to be located with the displays, not the shoeboxes. The protocol will exploit an interface such as Open GL or DirectX. This scheme will drastically cut down the amount of communications that must be provided by the MUX net and simplify technology refresh of the shoeboxes and the displays by co-locating technologies that are likely to be changed together.



SMART SKINS IN THE FUTURE



Outer Shell



Inner Shell


● Sensor, effector, strut

- *Tune EM & acoustic resonance*
- *Temperature, stress, & EM sensors*
- *Composite fiber network*
- *Skin is antenna*
- *Control air flow*
- *Genetically engineered materials*

Most possibilities not realistic for 2010 but embedded sensors might be. Improved health management via prognostics and envelop control.

Today probably limited to pasties on inner skin. Could still be a help.

19



Advances in material sciences, micro-miniature device technologies, and fabrication methods portend a dramatic change in the functionality of aircraft skins. Some of the possibilities include shaping to influence aerodynamic properties, built-in antennas and sensors, control of EM and acoustic signatures, embedded robust ship-wide communications media, and enhanced stress and temperature sensors for prognostics and diagnostics. While these prospects are exciting, most will not be ready for prime time in the next five to ten years. However, designers of new aircraft should revisit these opportunities frequently. These emerging technologies will provide new ways to increase performance and lethality while reducing form factor, weight, and costs. They should also ease the nightmarish problem of spotting so many pieces of odd-shaped equipment throughout the aircraft.



ENGINEERING CHALLENGES



The following slides discuss technology areas where engineering must be done in the near future to leverage scientific breakthroughs and even COTS products. The mere fact that a capability exists does not mean we can put it onboard a military air vehicle.

20

NORTHROP GRUMMAN

The previous slides have tried to present a picture of what future avionics might look like. The following slides identify work that is necessary to make that potential future a reality. In most cases the future portrayed only relies on COTS products, specialty products already available from specific sources, and published research results. In other words, the science contribution is in place. The engineering contribution, in many cases, is still pending. The necessary engineering is that which prepares technology for incorporation in military systems. The following slides summarize what must be done and provide baseline metrics and identify a few qualitative criteria that may be idiosyncratic to aircraft deployment.



SHOEBOX MODULES



SHOE BOX SPECS	
CPUS	5-20 Cores
Compute Power	50-200 Giga Instructions/Sec Total
RAM	10-300 Gbytes
ROM	500 Mbyte Non Volatile
Power Draw	100-200 Watts
Temperature	45-110 F
SLAN	Multiway THz +
Connections	3 Fiber LAN, 3 MuxNet
Modes	Tolerate/Isolate Core/LAN Failures
Robustness	Core+SLAN Net is 2-Edge Connected & 2-Node Connected



Find compromise MIL-STD that commercial suppliers can embrace. Need to increase volume in order to make it worthwhile for chip makers to keep DoD near leading edge of Moore's Law curve. Power PC & X86 chipsets.

21

NORTHROP GRUMMAN

The shoeboxes are the center of the avionics' computing capability. Each will resemble a multi-CPU server with the difference that it must be robust against physical damage. In particular, individual CPUs should be able to fail without compromising other components. The only direct connections to a shoebox are three high-speed fiber shoebox-to-shoebox connectors and three MUX net connectors. The net connections must be "multi ported" to the CPUs so that a single CPU failure does not delete a network connection. The shoebox must also have enough nonvolatile memory to boot the complete system and store data for system restarts and offline analysis. The total computing capability of each box should be in the 50 to 200 GHz range provided by 5 to 20 CPU cores and the total power dissipation should be no more than a few watts per GHz. It is reasonable to expect shoeboxes to run in controlled environments so extreme temperatures are not an issue.



MUX NETWORK



MuxBox Specs	
CPU	2 Cores
Speed	1-2 GHz Total
RAM	.25-1 Gbyte Toal
ROM	50 Mbyte Non Volitile
Communications	4 MuxNet, 20-40 Data, 2 Video
Modes	Tolerate/Isolate Core Failures
Temperature	?-250 F
Power Draw	50-100 Watts
Issue 1	Determine 2/3 Wire Data Standard
Issue 2	Data Network in High Heat

22

NORTHROP GRUMMAN

The MUX boxes and their connecting network will need to operate in sections of the aircraft where the environment is basically uncontrolled. Therefore, they must be robust to extreme temperatures and the like. Most of the connections to a MUX should be 2 or 3 wires with the latter able to supply moderate power to small devices such as temperature sensors. Each box should be able to connect 20-30 devices and the MUX should provide switchable A/D and D/A services to its clients. The MUX network must be compatible with the one used by the shoeboxes so that (1) the MUX network can provide alternative routing and (2) end-to-end protocols can be used. These features will provide system robustness in the face of MUX, MUX net, shoebox, and LAN failures. They will also greatly simplify implementation of avionics application code.



MULTILEVEL SECURE SYSTEMS



A challenge, not just a pretty picture

- In-theater COMM as well as profound reachback capabilities demands MLS
- MLS must be designed in - it is not an after-market add on
- Ramifications for software, hardware, systems, system-of-systems, and the operating policies and procedures
- Lack of MLS is number one threat to the success of network-centric warfare

Something must and will be done by somebody to plug this vulnerability. The only questions are who and when.

23

NORTHROP GRUMMAN

The emperor isn't wearing any clothes. We are constructing weapons systems for the network-centric world of the future yet few if any systems are certified for multilevel secure operations. How then are the various communications channels, rated anywhere from unclassified to top secret with tickets, going to connect to the same vehicle? The current solution is to operate system high and pass all information that must be downgraded through the pilot. In the network-centric future, this will subject pilots to unbearable workloads and automation will be precluded from assisting them. Multilevel security is a reasonably well-understood technology and it is possible to address these problems. The technology must be matured and fielded in a form that is usable for the many weapons systems that need it. Typical solutions use encapsulation and encryption in addition to policies and procedures that restrict the activities of automation and humans. The availability of copious computation will help address the technical issues: the use of a large number of CPUs will allow encapsulation by careful selection and control of execution sites and the large amount of computation power will permit small on-chip separate CPU cores to host encryption software. Many problems remain but it is time that the multilevel security problem be addressed.



HUMAN INTERFACE SPECS



DISPLAY TECHNOLOGY	
Coatings	Tolerant to G Forces, Field Refresh/Replace
Brightness Ratio	400-500 : 1
Isolation	Isolated Panel Failures
Electronics	2 Video Cards Per Panel, Feed from Mux Net
Power	45 Watts/Panel
Appearance	Seamless
Interface	Open GL (?)
Geometry	Curved
View	2- & 3-D No Goggles
Goggles	Track Eye Focus
Safety	Shatter Proof

The technology components to build reconfigurable cockpits already exist in the commercial world but that technology must transition to the military world to satisfy our needs. Some particular problems with display panels include ability to withstand G forces, shatter proofing, and fault and failure isolation. We have assumed that the equivalent of video cards are co-located with the displays, not the computers. This assumption reduces bandwidth between the displays and the computation suite and simplifies technology refresh. Some decisions must also be made as to how 3D imagery will be provided. One possibility is to use panel construction techniques which do not require special goggles. However, the full use of this strategy needs eye focus tracking in order to provide look-around-the-corner capability. In addition, a significant amount of human factors analysis is needed to decide how to best use the new possibilities that are now available.



SMART SKINS



SKIN TECHNOLOGIES	
Embedded Sensors	Stress, Temperature, EM
Interface	Mux Net Data Lines
Communications	Vehicle Net (?)
Countermeasures	DIRCM Pop Outs, etc.
Antennas	Embedded
Effectors	Embedded MEMs Shapers

A Wish List, Not Requirements

25

NORTHROP GRUMMAN

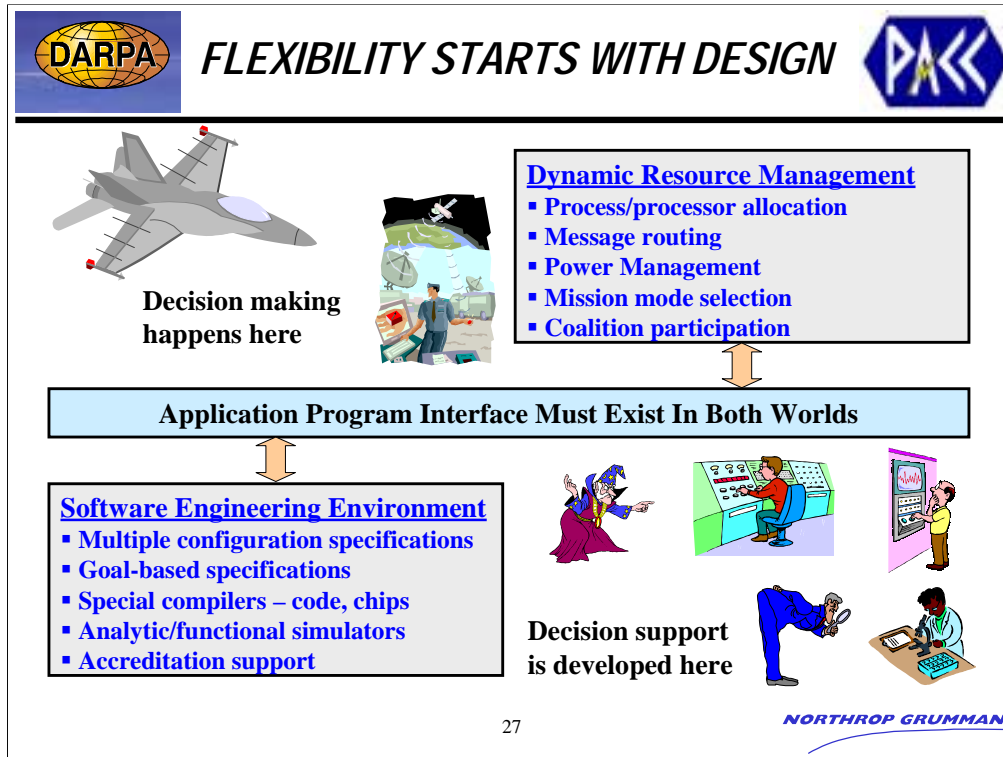
Smart skins technologies are probably not mature enough today to assure their existence or make specific plans for their use in the near-term future. Scientific breakthroughs, as noted previously, are still needed. The list on this slide represents some of the capabilities and problems that must be addressed for the first capabilities that will be matured in this area. Futuristic capabilities such as airflow shaping are mentioned also. One problem that needs immediate attention is the tradeoff between countermeasures and stealth. Countermeasures such as a DIRCM or a chaff dispenser need to “pop out” into the air stream in order to be deployed. Provisions for pop outs reduce stealth. Some research and new ideas are needed in this area.



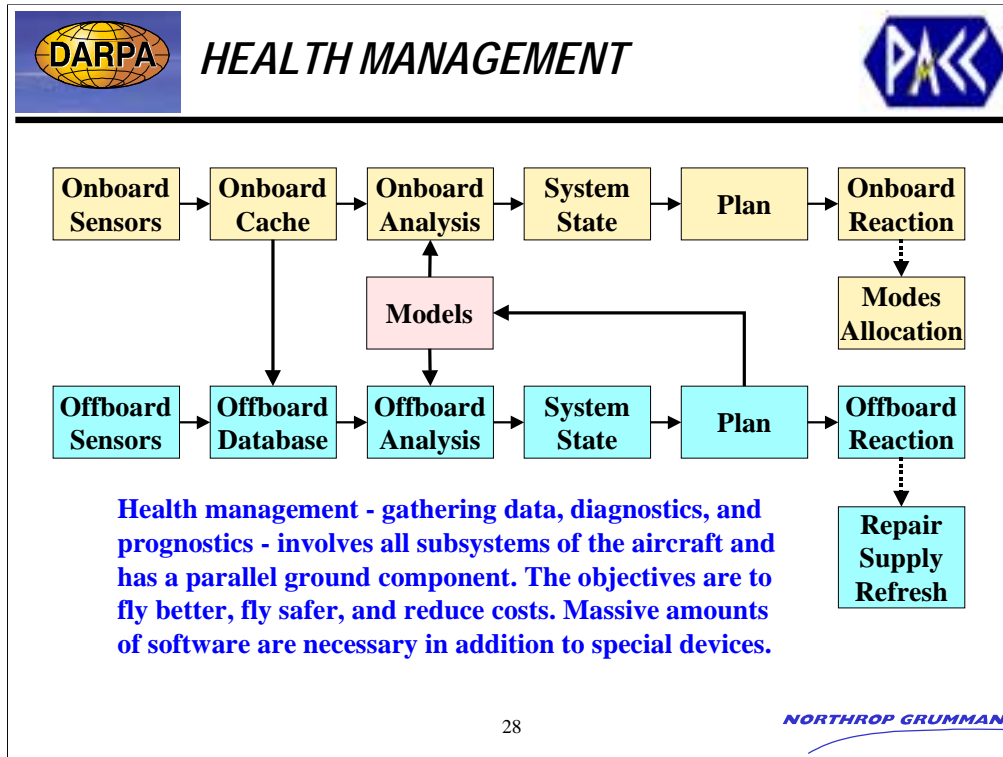
SOFTWARE TECHNOLOGIES	
Program Language Extensions	Interprocess & Data Dependencies
	Explicit Timing Assumptions
	Tolerable Slop Rates
Protocols	Specification Language (Ap-to-Ap)
Code Sources	Mat Lab Models -> Flyable Code
Timing	Delta t from Time Stamp, Not a Constant
Documentation Source	Auto Generation from Code & Spec
Standards	Compatible V&V, Accreditation, Certification
Control	Compatible Configuration Control
Process	Compatible Process Tailoring

More concepts and details in following section


Software is the biggest cost factor in avionics and avionics is projected as the largest single cost element in the aircraft lifecycle. Better ways are needed to design, implement, document, maintain, and upgrade avionics software. The suggestions made herein include process-centric architectures, end-to-end protocols, more flexible time management, and better coding tools. In today's production model, avionics programs are implemented three times: in the design, in the detailed specifications, and in the code. The engineers in the first two phases actual go through a type of coding exercise to debug their specs and then document the results. The process is reminiscent of an old math joke: the mathematician guesses a theorem with a stroke of insight. By the time its published, the form is bottom-up deductive reasoning with no trace left of the insight. The documentation passed from requirements to the detail specifiers and from them to the programmers recapitulates this loss of insight. The use of tools that go from mathematical specifications to code is one way to remove the middleman and capture more design intent. Further, the use of programming languages where forms capture intent as well as procedural specifications would help a lot. In addition, there are ways to simultaneously generate documentation and code. All of these ideas have been around for a long time and a few military programs are starting to use them. However, it behooves us to promote better tools and methods. In addition, we must rethink our processes, procedures, and configuration control methods so that they are appropriate to the new way of doing things.



The power and flexibility planned for future avionics will need new methods to support design and development. Dynamic process/processor allocations, power management, and message routing in addition to flexible mission management, coalition participation, and more autonomous software will stress the de facto methods used today. It is too complicated to have each subsystem factor these considerations as well as the demands of realtime computations. The software engineering environment of the future will need to incorporate new tools and languages to support sensible implementation, benchmarking, and accreditation strategies. Another problem is how to communicate the knowledge gained in the SEE to the runtime system where it is used. The application program interface shown in the figure is a representation of a mechanism to do that. The design, coding, and analysis tools in the SEE must automatically provide information that is either compiled into or delivered to the runtime system where it is used to improve the performance of the system and ensure that proper behavior results. There are several current research and development efforts that support this model, e.g., the Morphware Forum within the DARPA IPTO PCA program. It is difficult enough to incorporate a single new paradigm, e.g., natural autonomy or power management, into a realtime system. Since the proposal is many new paradigms, careful thought is necessary since the problem is orders of magnitude more difficult. Both software and hardware engineers have made great strides in the last decade to develop tools that support the rapid development of complex systems. For our vision of the future to succeed, that progress must continue.




The goal of health management is to prevent failures as well as recover from failures during flight. The objectives are to fly better, fly safer, and reduce lifecycle costs. Virtually every subsystem and every component can fail or be compromised by physical assault. Onboard diagnostic systems must detect the failures and provide alternative resources that preserve flight. Prognostic systems must analyze usage data and predict potential failures so that they can be prevented. Onboard prognostics can change mission plans. This slide shows a normative health management system. That system consists of sensors to detect failures and capture usage data, components to accumulate data and determine state, and components that select alternative solutions or implement repair strategies. The ground system parallels the onboard system. Both are comprised of massive amounts of software: data management facilities, model-based reasoners, and capabilities to interact with the flight and ground crews. Much of the software is hidden within OFPs so the true size of the resources devoted to health management is not always appreciated. Future avionics implementations must improve performance and control costs through better technology. The architecture promoted herein will benefit that cause because it enables the rational development of a distributed health management paradigm where more information can be shared, better decisions can be made, and the impact of technology refresh activities can be better analyzed. More research is needed in the development of sensor devices to support health management and to build better models of the effects of usage on health.



DARPA


GIFT WRAPPING



- *Composites and many other light weight, strong materials available*
- *Current racks, packaging, extremely heavy and bulky.*

Goal: reduce total weight by 500-1000 lbs & form factor by 2 using new materials.

A commercial example today only 25 lbs.



Lightweight composite rack with three shelves to three HF/VHF/UHF SATCOM type transceivers

Bottom of unit will be used for AC to DC Power Supply shelf area will be used for either a wired or wireless system

Three middle shelves will be used for the transceiver with space available for a SATCOM/Line-of-Site RF Antenna

Space for portable computer for system control


Self contained system will allow cabling to be contained within the rack close out panels.

Lightweight composite rack without equipment is 25 pounds

System can be operated from AC or DC power source for worldwide applications, vehicle installations, aircraft installations or

Racks can be configured for different form factors to fit particular applications. Please contact McDowell Research for specific requirements

29



The inside of an aircraft is a rather exotic volume with a strange shape. It's what's left over after aerodynamics, propulsion, and structures have had their say. Spotting avionics gear has a tradition of placing square pegs (rectangular boxes) in round holes. It is possible to invent manufacturing techniques that can produce lightweight packaging that will actually fit the available spaces. With the vast reduction in the size of electronics components, future mil spec boxes are going to be rather empty. If an assault on form factor is mounted, I believe that the volume consumed by avionics could be reduced by a factor of two. Whether that path is pursued or not, it should be possible to achieve significant weight reductions by simply constructing racks and boxes out of lighter materials. The inset picture in the slide shows a composite rack that weighs less than 25 lbs. and holds three SATCOM receivers. With some attention to manufacturing methodology, particularly automated ways to go from CAD databases directly to product build instructions, there should be a cost-effective path to significant savings in form factor and weight.



ACTION ITEMS



There is an opportunity and lots to do. The aerospace industry must position itself to profit from the future. DARPA should be encouraged to invest in this vision and we should do everything possible to leverage the available technology to build better, more cost-effective weapons systems.

Contractor Items

- *Understand the opportunity*
- *New integration strategies*
- *Coherent picture of future*
- *Plans for that future*

Government Items

- *Evaluate scientific progress*
- *Assess engineering needed*
- *Establish engineering agenda*
- *Move industry into the future*

30

NORTHROP GRUMMAN

The previous slides describe several engineering challenges that will enable better, more cost-effective avionics systems in future aircraft. Some of the necessary engineering will happen through momentum. But not all of it. We suggested that experts in various domains extend the list started here and do a more thorough job of filling in the requirements. Further, it needs to be shown, in detail, that these requirements support our future avionics vision. The next step is to inform DARPA and the joint services of the need to transition research that they have sponsored into future weapons systems. The basic science is in place and it is time to figure out how to best exploit the results.



SOFTWARE ISSUES



The next group of slides discuss various software issues. Software is the Achilles heel of avionics - In the long run it costs more than the hardware, it is the chief source of vulnerabilities and errors, and it is the most complex component of a modern aircraft.

31

NORTHROP GRUMMAN

The next set of slides amplify some of the major issues that impact the design, development, maintenance, and upgrade of software. The topics are timing issues, local (onboard) communications, determining and tracking system-wide data dependencies, qualitative guarantees of safety and liveness (the latter are technical terms that describe features of process executions), and vehicle-wide health management. The major stress on the timing and scheduling of avionics code is the flight control loop. The specific requirements that result from this source are discussed.



RIGID TIME-HANDLING DISASTER



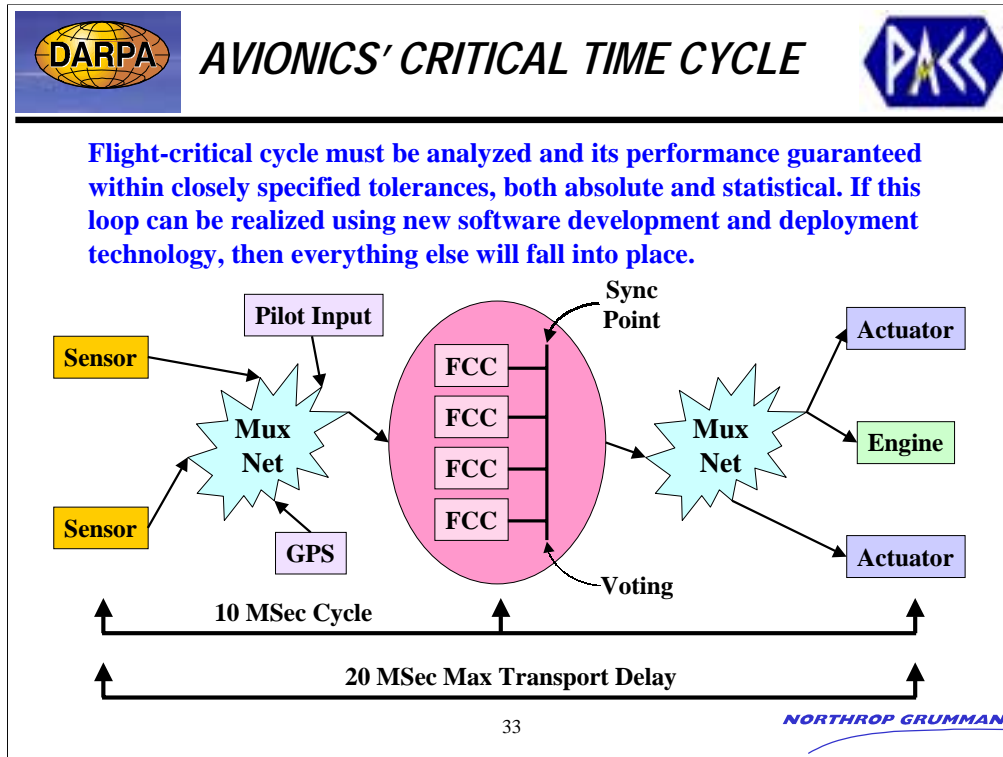
- ***Complexity of avionics is driven by size, lack of resources, and timing issues.***
- ***Insufficient resources cause the most problems.***
- ***Resulting time-management strategy includes:***
 - *Time constants that are compiled into the code*
 - *Redundancy voting schemes must compare data from the same epoch*
 - *Careful hand coding of complicated mathematics*
 - *Complete offline scheduling of computation/communications*
 - *ICD that compresses low-level timing and storage issues with high-level dataflow and data type specifications.*

Better Methods are Sorely Needed

32

NORTHROP GRUMMAN

Exacting timing requirements coupled with lack of adequate computing and communications resources has lead to a style of system development where code is difficult to write and validate. The style typically used makes it hard to determine which implementation decisions are requirements driven and which ones are not. It is also difficult to determine dependencies within the system. Therefore, when it comes time to change things a sizable resource investment is needed to determine what all will be impacted. Several devices in the system have specific timing requirements, e.g., control surface actuators, the display refresh logic, and small sensors whose output must be latched within a narrow time window. However, most avionics software is written as if it had exacting timing even if it doesn't touch these devices and isn't on a critical path between them. This is done so that the timing of adjacent executions of time-critical components can be guaranteed. Lack of resources has made this brittle scheme necessary. In addition to exacting offline scheduling of process executions, communications is scheduled in the same manner: timeslot reservations are made at design time for buses and permeate the entire design. Thus a change to a processing requirement must deal with the impact on the execution schedule of the processor where the change will be implemented and the communications timing of everything on the buses used for input/output to that processor must be revalidated. In addition, the interface control documentation (ICD) must reflect all of these changes. Better software technology can have a major impact on the situation.



This slide depicts the time-critical flight-control loop. Much of the detail has been omitted. The sensors measure attitude, acceleration, etc., and pass that data to the flight control computers in the main computation suite where the data is filtered and fused with GPS data and commands from the pilot or auto pilot. The solutions generated by the redundant FCC computers are compared and voted upon. The selected solutions are then forwarded to the actuators and the engine controller. Data transport to and from the main computation suite is via the MUX net. The commands given to the actuators should be based on data that is not more than 20MSec old. Further, new commands should arrive at actuators every 10MSec to effect a 100Hz control rate. This means that the sensors as a group, the control computers as a group, and the actuators must cycle every 10Msec. The major synchronization point shown in the figure must be precisely controlled in the order of 1MSec. We can then do a requirements allocation of the maximum processing time allowed to each component set shown in the diagram including the MUX net in order to stay within the 20MSec staleness requirement. Something that isn't often realized is that no harm would be done if there is an occasional mistiming at the actuators. For example, if data was unduly delayed as much as once or twice a second no loss of control would be evident. The point is that, strong statistical guarantees would be good enough in many cases. The acid test for the software avionics organization proposed in this briefing is whether it can adequately handle the loop shown here.



BETTER TIME MANAGEMENT




- *Resource availability is growing faster than avionics complexity*
- *Even flight-critical code can tolerate occasional timing violations on the input side*
- *So*
 - *More use of filters to combine data; use models to predict next output when necessary*
 - *Use data time stamps rather than constant δt*
 - *Generate code from math (E.G., MATLAB), not Jovial or C*
 - *Timing requirements applied dynamically*

The resulting system is less costly, easier to design, maintain, and mature, and much more robust.


34

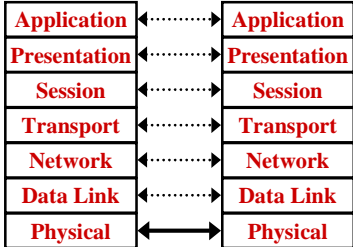
NORTHROP GRUMMAN

There are many ways to meet the timing requirements imposed by the flight control loop discussed on the previous slide. However, it would help to learn what those requirements actually are. In the past, resource limitations led to the preplanned periodic cycle of the entire avionics suite and a corollary of that approach: If you miss a deadline once, you will miss it again and again. The fact that a deadline occasionally could be missed without harm was, thus, irrelevant. Since the premises are no longer true, neither is the conclusion. As noted previously, we can trade copious resources to construct a better, more flexible system where implicit timing dependencies are weaker. This fact will reduce the costs - time and money - necessary to design, build, maintain, and refresh avionics systems. This slide lists a few of the techniques that can contribute to such an approach. Note, they all consume additional resources. One issue not discussed here is the need to rethink requirements that must be placed on flight controls and how we validate, verify, and accredit future systems. New methods are needed. We point out that this proposal is not the most radical one under current consideration. For, example, DoD labs are sponsoring research in the use of dynamically adapting neural-net flight controllers.



MODERN COMMUNICATIONS







**Protocol Stack
Architecture**

Avionics issue is the selection/design of sufficient stack architectures to delivery necessary range of QoS guarantees. Much of the necessary aircraft accreditation and certification strategy will flow from arguments about the communications structure and its implementation.


- End-to-end & multicast services in face of process relocation
- Different connections need different Quality of Service (QoS)
- Different stacks may be built on same media and single stack may use multiple media
- Stack levels may be compressed

35


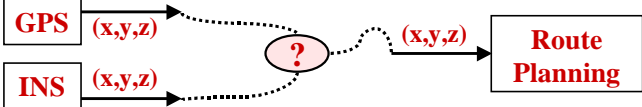
A protocol is a set of rules that govern a potential communication or dialogue. Higher level protocols are built on others, e.g., a FAX protocol is built on the phone protocol with its rules of dialing, information exchange, and termination without knowing how the phone system works. Modern communications are built out of protocol stacks as shown on the slide. Note that a single layer can support multiple high-level protocols, e.g., the phone supports data terminal and voice in addition to FAX. Further, a single protocol can be supported by different stacks, e.g., a FAX can be transmitted using the Internet. The point of these remarks is that different protocols can coexist using shared mechanisms. This fact is important to avionics because different protocols typically offer different qualities of service and different guarantees. In most current avionics solutions a single protocol built on the 1553A is used for all communications. In future systems, interprocess communications will be designated on a case-by-case bases and the necessary quality of service and guarantees will dictate the choices. Flight controls will surly make a different selection than weather-data distribution. The current solution is called host-to-host (or processor-to-processor) connectivity. It is the host's responsibility to distribute the data to the correct applications. In the approach we propose, end-to-end (application-to-application) methods will be used. The advantages are the ability to relocate processes for load balancing and error recovery and a much sharper map of applications dependencies. Thus, the resulting system will be more robust and technology refresh will be simplified and less costly.



DEPENDENCY MANAGEMENT



- *Dependency management is key to sane maintenance and technology refresh.*
- *Provides answers to the questions*
 - If I change X will Y be effected?
 - Who should I consulted before making the proposed change?




```


graph LR
    GPS[GPS] -- "(x,y,z)" --> Q((?))
    INS[INS] -- "(x,y,z)" --> Q
    Q -- "(x,y,z)" --> RP[Route Planning]
  
```

- *Does RP depend on GPS or INS? Can't tell by variable names.*
 - Might be using GPS, INS solution, raw data from another source, or some fused version.
- *Requirements tell us what SHOULD be the case but protocol will tell us what is today's truth.*
- *Provides another source of configuration control information.*

36



Understanding data dependencies is the key to understanding a large software system. That knowledge allows us to judge whether the system components have the right information at the right time as well as the potential impact of modifying data sources. Such judgements in today's systems are difficult for two reasons: The first is that multiple logical messages are combined to save scarce communications resources and the ICD does not distinguish utilization below the OFP level. The second reason is there is little distinction between an OFP acting as a store-and-forward device for a message and using its contents. The end-to-end protocol techniques and fast networks advocated herein will address both of these issues. Message protocol specification exactly identify the producers and consumers of all data items. Transport, routing, and delivery mechanisms as well as multiplexing and demultiplexing strategies are private concerns of lower level protocols and do not, in any way, effect the logical dependencies of the system. Therefore, what is documented is the truth about the executing system. In fact, the equivalent of an ICD can be auto-generated from protocol descriptions rather than vice versa. The improvements in system analysis tool capabilities and system understanding that will result therefrom should provide substantial reductions in cost and time throughout the entire avionics lifecycle.




QUALITATIVE SYSTEM ATTRIBUTES



- ***Timing qualities besides promptness are needed:***
 - Safety properties: deadlock, mutual exclusion, termination
 - Liveness properties: fairness, message receipt, message send, eventual service
- ***Tools can check these properties from abstract dependency models***
- ***Lifecycle strategies using them are needed***



37



Systems such as avionics that provide for multiple simultaneous execution of dependent processes are difficult to analyze. Critical timing dependencies are an example previously discussed. Other potential problems are divided into two categories: safety and liveness. Safety concerns the prevention of unrecoverable computational states such as deadlock, two or more processes entering a critical region together, e.g., to update the same data simultaneously, or a process entering a non-terminating loop. Liveness concerns all processes receiving a reasonable share of services and the system, as a whole, making progress towards its computational goals. Guaranteeing these properties is difficult because they depend on global system composition and the rules governing its behavior while designs are centered on local interactions. Many tools exist to model concurrent systems and analyze specific properties. There is also a large research and COTS community to support and improve the tools. It is recommended that future avionics systems use these products for both initial design and technology refresh activities. It is important that their use be codified in the policies and procedures that control the software lifecycle.



Power-Aware Challenges for Unmanned Air Vehicles



Jeffrey A. Barnett & Rich Ramroth
Northrop Grumman Corporation

For the
Software Radio Group
June 20, 2002

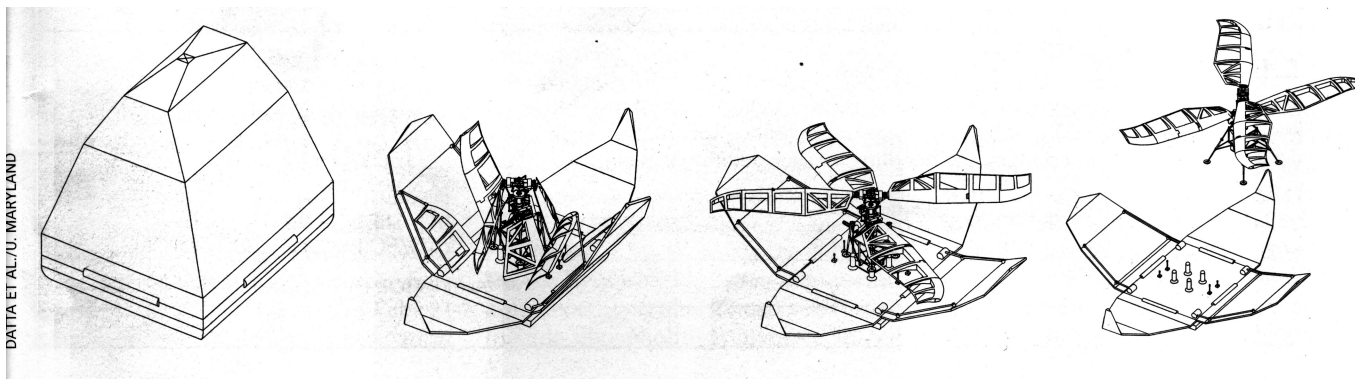




Power-Aware Challenges for Unmanned Air Vehicles



- Air design and operations are always a tradeoff in the (FORCE x ENERGY x PERFORMANCE x COST) space
- Integrated solutions are needed - attacking a single point is like pushing on one side of a marshmallow - interactions will be discussed below
- Tradeoff space is too complex for precise mathematical models - heavy reliance on simulation
- Application value system is King; less isn't always better





The Size of Things



Features	Min Values	Max Values
Weight	ounces	tons
Wing Span	inches	200+ ft
Endurance	seconds	days
Power Source	(solar) battery	generator
Propulsion	none	engine
Communications	none	multi network
Sensors	none	gps, self, outside
Controller	remote pilot	autonomous
Socialization	loner	swarm
Life Time	seconds	multi mission
Cost	\$x00.00	\$x0,000,000





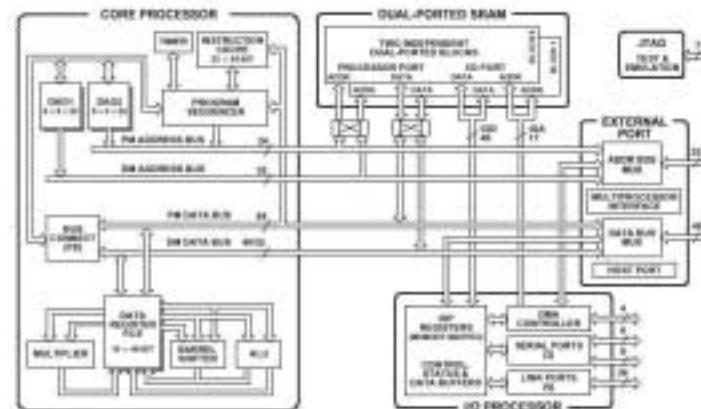
Key Technology Areas



- **Computational Challenges**
- **Communications Challenges**
- **(Non-Reflexive) Sensor Challenges**
- **Flight Controls Challenges**



- **Mission Planning and the Virtual Pilot**
- **Prognostics and Diagnostics:** If use slack time, no chance to enter low-power modes; use of internal sensors and BIT
- **Redundancy Management:** N-way redundancy means N^+ times the energy expenditure
- **Scheduling:** Computation gaps enable low power; slow computation uses less power; how many CPUs?



Primary interest to Software Radio Group

More Than Cute Protocols Are Needed

- **Economical Listening:** listening mode usually is as expensive as receiving mode
- **Message Encoding:** encryption/compression are not commutative; both defeat economical listening
- **Distance:** cost is related to sender/receiver distance so path planning should be considered
- **Good of the Group Vs Individual Payoff**
 - **Two for One:** savings through multiplexing and/or message routing must be considered. Broadcasting tradeoffs?
 - **Silence is Golden:** when is it better to communicate and when is it better to say nothing? Error rate Vs Power.



Primary interest to
Software Radio Group

- **Choices: which sensor to use;**
 - use an ownship sensor
 - communication to borrow another's data
 - tease more info by computation
- **Aiming: gimbal the sensor VS fly a different heading**
- **Internals: sensor boxes comprise vast computation arrays**
- **Power: variance reduction is function of resource investment, both rate and pulse strength**
- **Image Processing: computational investment before filtering and fusion**
- **Filtering and Fusion: incremental value of better algorithm**



Primary interest to
Software Radio Group



Flight Management Challenges



- **Flight Controls:** actuators use electric power; tradeoff between power and maneuver pace
- **Reflexive Sensing:** sensing ownship's status is part of feedback and control mechanism; adds local communication and computation costs
- **Multiple Vehicles:** coordination consumes communications resources and shake-and-bake control consumes energy



**Primary interest to
Software Radio Group**