DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

nk)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	10/93	Interim 12/01/92 - 09/30/93

4. TITLE AND SUBTITLE
Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science

5. FUNDING NUMBERS
N00014-93-1-0152

6. AUTHOR(S)
Scott B. Baden

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Regents of the University of California
University of California, San Diego
Dept. of Computer Science & Engineering
9500 Gilman Dr., La Jolla, CA 92093-0114

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Scientific Officer Code: 1133
Gary M. Koob
Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
(a)

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
No Limitations

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

14. SUBJECT TERMS	15. NUMBER OF PAGES
	29
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

UNIVERSITY OF CALIFORNIA, SAN DIEGO

UCSD

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO

SANTA BARBARA • SANTA CRUZ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING, 0114
FAX: (619) 534-7029

9500 GILMAN DRIVE
LA JOLLA, CALIFORNIA 92093-0114

October 4, 1993

Chief of Naval Research
Code 3330/Annual Report
Ballston Tower One
800 North Quincy Street
Arlington, VA 22217-5660

Dear Dr. van Tilborg,

Enclosed you will find the annual report for "Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science," contract number N00014-93-1-0152. This report covers the period 1 Dec 92 through 30 Sep 93 (the contract began 1 Dec 92), and includes a recent publication to accompany the briefing section.

Sincerely Yours,

Scott B. Baden

Scott B. Baden
Assistant Professor
(619) 534-8861

UNIVERSITY OF CALIFORNIA, SAN DIEGO UCSD

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO SANTA BARBARA • SANTA CRUZ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING, 0114
FAX: (619) 534-7029

9500 GILMAN DRIVE
LA JOLLA, CALIFORNIA 92093-0114

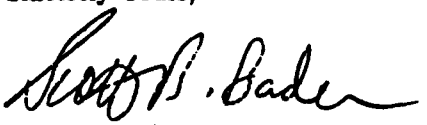
October 4, 1993

Chief of Naval Research
Code 3330/Annual Report
Ballston Tower One
800 North Quincy Street
Arlington, VA 22217-5660

Dear Dr. van Tilborg,

Enclosed you will find the annual report for "Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science," contract number N00014-93-1-0152. This report covers the period 1 Dec 92 through 30 Sep 93 (the contract began 1 Dec 92), and includes a recent publication to accompany the briefing section.

Sincerely Yours,



Scott B. Baden
Assistant Professor
(619) 534-8861

UNIVERSITY OF CALIFORNIA, SAN DIEGO UCSD

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO SANTA BARBARA • SANTA CRUZ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING, 0114
FAX: (619) 534-7029

9500 GILMAN DRIVE
LA JOLLA, CALIFORNIA 92093-0114

October 4, 1993

Dr. Robert Voigt
New Technologies
National Science Foundation
1800 G Street NW
Washington, DC 20550

Dear Bob,

I am enclosing a copy of my ONR annual report for "Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science," ONR contract number N00014-93-1-0152. I am required to submit this report to you because the award period of the ONR contract (12/1/92 to 11/30/95) overlaps that of my NSF Research Initiation Award, Number ASC-9110793 (7/1/91 to 6/30/93). The annual report includes a recent publication to accompany the briefing section.

Copies of this report have been sent to the following individuals:

Chief of Naval Research
Code 3330/Annual Report
Ballston Tower One
800 North Quincy Street
Arlington, VA 22217-5660

and

Dr. Gary Koob
Computer Science Division
Code 1133SS
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5000

Sincerely Yours,

Scott B. Baden
Assistant Professor
(619) 534-8861

Annual Progress Report
October, 1, 1993

Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science

Contract N00014-93-1-0152

Scott B. Baden
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

(619) 534-8861
baden@gili.ucsd.edu

Principal Investigator:	**Scott B. Baden**
Institution:	**University of California, San Diego**
Phone number:	**(619) 534-8861**
E-mail address:	**baden@cs.ucsd.edu**
Contract title:	Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science
Contract number:	N00014-93-1-0152
Reporting period:	1 Dec 92 - 30 Sep 93

Contents

Principal Investigator:	**Scott B. Baden**
Institution:	**University of California, San Diego**
Phone number:	**(619) 534-8861**
E-mail address:	**baden@cs.ucsd.edu**

Contract title:	**Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science**
Contract number:	**N00014-93-1-0152**
Reporting period:	**1 Dec 92 – 30 Sep 93**

1 Numerical productivity measures

Refereed articles in preparation: 4
Refereed papers published: 1
Unrefereed reports and articles: 1
Invited presentations: 7
Contributed presentations: 1
Graduate students supported >= 25 % of full time: 2

Principal Investigator:	Scott B. Baden
Institution:	University of California, San Diego
Phone number:	(619) 534-8861
E-mail address:	baden@cs.ucsd.edu
Contract title:	Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science
Contract number:	N00014-93-1-0152
Reporting period:	1 Dec 92 - 30 Sep 93

2 Summary of technical results

The purpose of this investigation is to explore software techniques for reducing the development costs of scientific applications running on distributed memory MIMD parallel computers. Our objective is to develop a domain specific software development tool called LPAR. LPAR is targeted to non-uniform numerical methods that employ elaborate representations—such as adaptive and multilevel finite difference meshes or particles—to accurately capture irregular phenomena such as turbulence and shocks.

We are using LPAR to develop new parallel software techniques and numerical algorithms for the predictive modeling of technologically important materials using first principles simulations. This is a multidisciplinary collaboration and involves four other investigators—Alan Edelman (MIT, Mathematics); Ryoichi Kawai (U. Alabama, Birmingham, Physics); Beth Ong (UCSD, Mathematics); and John Weare (UCSD, Chemistry).

A prototype of LPAR is running on the Intel iPSC/860, nCUBE/2, and on workstations. Applications written with LPAR are portable across diverse computer architectures and qualitatively simpler than hand-parallelized software, e.g. with explicit message passing. Moreover, this simplicity incurs only a modest overhead: applications running on the Intel and the nCUBE perform competitively with implementations running on a single processor of the Cray Y-MP. Because LPAR software is machine independent, we are able to develop parallelized applications on workstations; programming environments on these architectures are generally more mature than on parallel hardware platforms.

LPAR provides dynamic memory management facilities that are unavailable in conventional C and Fortran-based programming environments. LPAR enables the programmer to manage locality explicitly and this contributes to the low operating overheads observed in practice. LPAR's dynamic memory management facilities may coexist with existing programming languages. Our investigation suggests a plausible way to enhance such languages to adequately treat irregular scientific applications.

LPAR is based in part on the scientific programming language FIDIL[1]. It extends the simple array model of Fortran-90 to support dynamic, non-uniform, physically distributed arrays in a shared name space. LPAR supports coarse-grain, irregular array decomposition and enables the user to manage locality explicitly in order to limit overhead costs such as communication. It provides high-level communication mechanisms and thus frees the programmer from low-level details involved in partitioning, synchronization, and communication. Because LPAR hides architecture-dependent details, applications are portable across a wide range of parallel computers with little loss in performance.

[1] P. N. Hilfinger and P. Colella, "FIDIL: A Language for Scientific Programming," Lawrence Livermore National Lab. Tech Report UCRL-98057, January, 1988.

LPAR efficiently supports irregular computational domains comprising meshes organized into levels. These are represented with a single data structure, known as the Map, which is a dynamic array with an irregular index set. A finite difference grid is represented in LPAR as a Map, and a level of a mesh hierarchy is represented as a Map of Map. A map of maps is intended to be decomposed at a coarse level of granularity; typically, only a few map components are assigned to a processor. Each map element is assigned to a single processor, a process which is a coarse-grain analogue of Fortran D's irregular array decompositions. Algorithms for decomposing problem domains and assigning maps to processors are application-specific and are external to LPAR. LPAR provides a common framework for specifying the decompositions and processor mappings, as well as a standard library that includes commonly used partitioning and mapping routines.

Associated with each map is a quantity known as a Domain. A domain represents the index set of a map and is a first class object that may be assigned and manipulated using intrinsic *domain calculus* operations such as intersection. Since maps may have irregular domains—with ragged edges and holes—the domain concept generalizes the Fortran-90 array section.

LPAR provides coarse-grain looping constructs to express concurrent execution. Regions of data dependence among the elements of a Map of Maps are described geometrically—as intersections between maps—and dependence data is copied using assignment. Thus, although LPAR requires the programmer to explicitly manage parallelism, it significantly raises the semantic level of this activity to a manageable level.

LPAR uses a uniform representation for all data—the map—and a uniform representation for structural information—the domain. Experimentation has revealed an advantage to distinguishing Maps of scalars (e.g. real numbers) from Maps of Maps, as well as distinguishing rectangular from irregular domains. There are two major advantages for making these distinctions: a reduction in run time overheads, and a simplified implementation. We have incorporated these ideas into LPAR, and call the resulting system LPARX. Irregular domains have been omitted, since the applications we are are interested in require only rectangular domains. However, the definition of LPARX naturally admits irregular domains without fundamentally changing the programming model.

LPARX replaces the LPAR Map with two new data types: Grid and XArray. A Grid is a restricted form of an LPAR Map, which has square domain. the XArray construct replaces the Map of Map. Parallel calculations are represented as an XArray of Grid. The XArray is used to represent the parallel structure of a computation: there is a coarse grained looping construct defined over XArrays and the Grids comprising an XArray communicate using higher-level copy operations.

The LPARX implementation is based upon an object-oriented parallel programming library called Distributed Parallel Objects (DPO). DPO provides high-level abstractions for manipulating physically distributed objects in a shared name space. It provides facilities for asynchronous inter-object communication, synchronization, data redistribution, and automatic load balancing. DPO enables the rapid implementation of efficient object-oriented parallel code.

LPARX and DPO have been implemented as C++ class libraries and are currently running on the CM5 (Los Alamos National Laboratory and Naval Research Laboratory), Intel Paragon (San Diego Supercomputer Center (SDSC)), nCUBE/2 (SDSC), KSR (Cornell Theory Center), workstations, and networks of workstations under PVM. We are currently implementing a variety of applications, including: two- and three-dimensional particle dynamics, multigrid and conjugate gradient solvers for elliptic systems, and adaptive mesh refinement for solving hyperbolic conservation laws.

To assist in our evaluation of LPARX, we are developing a subset of our applications under the fine-grained Fortran-90 programming model. Because we require precise control over low level details such as layout and the generation of temporaries, we have implemented a subset of Thinking

Machine's CM Fortran as a C++ class library. A prototype system—called PMPL— is running on the nCUBE/2. A portable production version is currently under development, and will be installed on the Paragon.

Complimenting the above software investigation is a collaboration with our multidisciplinary research group to develop fast Poisson solvers for application to first principles simulations. We have developed 3-D surface charge methods for rapidly computing the Hartree potential in real space. This is a prerequisite step toward the development of adaptive mesh methods for efficiently representing electronic wavefunctions over multiple length scales and open structures such as C_{60}. (Currently, Fourier space representations are employed.)

Principal Investigator:	Scott B. Baden
Institution:	University of California, San Diego
Phone number:	(619) 534-8861
E-mail address:	baden@cs.ucsd.edu
Contract title:	Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science
Contract number:	N00014-93-1-0152
Reporting period:	1 Dec 92 - 30 Sep 93

3 Lists of publications, presentations and reports

Publications

1. S. B. Baden and S. R. Kohn, "Portable Parallel Programming Under the LPAR System," to be submitted for publication, October 1993.

2. S. R. Kohn and S. B. Baden, "An Implementation of the LPAR Parallel Programming Model for Scientific Computations," in *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, Norfolk, Virginia, 1993, pp. 442–450.

3. S. B. Baden and S. R. Kohn, "Lattice parallelism: A Parallel Programming Model for Manipulating Non-Uniform Structured Scientific Data Structures," *ACM SIGPLAN NOTICES*, January, 1993, pp. 24–27.

Presentations

1. San Diego Supercomputer Center Steering Committee Meeting; Dec. 14, 1992; Title of Presentation: "System support for scientific computations."

2. UCSD Connect "Meet the Researcher Series;" January 15, 1993; Title of presentation: "Considerations in Using Massively Parallel Computers: A Minimalist Approach to Writing Robust Software."

3. University of Maryland, Computer Science Department; March 26, 1993; Title of Presentation: "The LPAR Programming System for Parallel Computation of Scientific Problems."

4. University of California, Berkeley, Computer Science Division; April 30, 1993; Title of Presentation: "The LPAR Programming System for Parallel Computation of Scientific Problems."

5. University of Wisconsin, Madison, Computer Science Department; June 28, 1993; Title of presentation: "Programming Scientific Calculations with LPAR."

6. MetaCenter Computational Science Institute in Parallel Computing, San Diego Supercomputer Center; Aug 11, 1993; Title of presentation: "Programming Scientific Calculations with the LPAR Programming System."

7. Lawrence Livermore National Laboratory; Aug 13, 1993; Title of presentation: "Programming Scientific Calculations with the LPAR Programming System."

Principal Investigator:	Scott B. Baden
Institution:	University of California, San Diego
Phone number:	(619) 534-8861
E-mail address:	baden@cs.ucsd.edu

Contract title:	Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science
Contract number:	N00014-93-1-0152
Reporting period:	1 Dec 92 - 30 Sep 93

4 Description of research transitions and DoD interactions

We are also collaborating with Dr. Bill Crutchfield and Dr. Chuck Rendleman of the Applied Mathematics Group at Lawrence Livermore National Laboratory (group leader: Dr. John Bell) to develop portable adaptive mesh refinement software for 2-D and 3-D gas dynamics. The Livermore group is adopting LPARX in order to avoid the proliferation of architecture dependent code streams.

The research also involves interactions with scientists at Los Alamos National Laboratory, in particular a group under the supervision of Dr. Jeffrey Saltzman and Dr. David Brown. The Los Alamos group is developing an array class library, called P++, for adaptive mesh methods. Together, LPAR and P++ are helpful in simplifying the implementation of complex, adaptive mesh refinement algorithms for the study of applications such as the integrated circuit trenching problem.

Principal Investigator:	Scott B. Baden
Institution:	University of California, San Diego
Phone number:	(619) 534-8861
E-mail address:	baden@cs.ucsd.edu
Contract title:	Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science
Contract number:	N00014-93-1-0152
Reporting period:	1 Dec 92 – 30 Sep 93

5 Description of software and hardware prototypes

LPAR and LPARX are implemented as C++ class libraries. The LPAR prototype is available via anonymous ftp. A beta test version of LPARX will be released in November, 1993 (via anonymous ftp), and a production version also distributed to the NSF Metacenters through our local contact with the San Diego Supercomputer Center. The Distributed Parallel Object class library (DPO) will also be made available along with the `MP++` class library which supports stream-oriented, architecture-independent message passing facilities. (LPAR is written on top of DPO, which is in turn written on top of `MP++`.)

Principal Investigator:	Scott B. Baden
Institution:	University of California, San Diego
Phone number:	(619) 534-8861
E-mail address:	baden@cs.ucsd.edu
Contract title:	Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science
Contract number:	N00014-93-1-0152
Reporting period:	1 Dec 92 – 30 Sep 93

6 Briefing material

Please see attachment on following pages.

Parallel Programming Methodologies for Non-Uniform Structured Problems in Materials Science

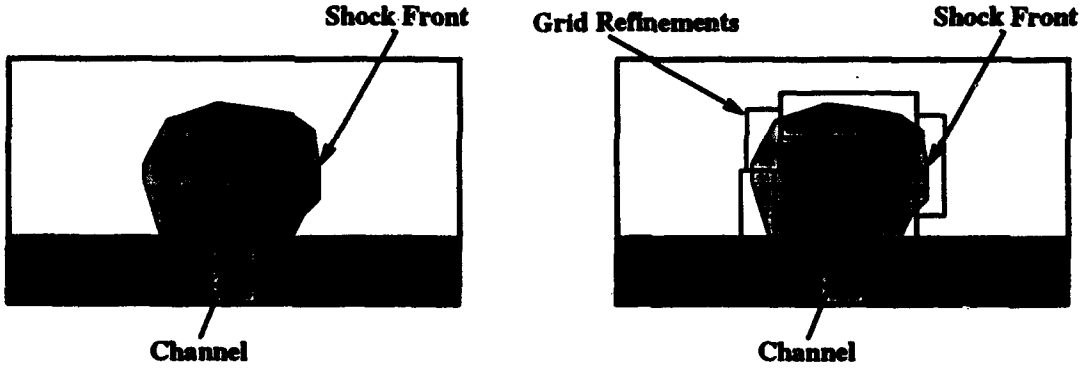
Scott B. Baden

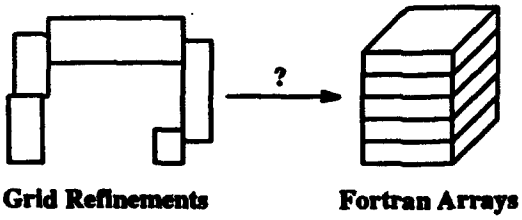
Computer Science and Engineering
University of California – San Diego
La Jolla, CA 92093 – 0114

Contract number: N00014-93-1-0152

Motivation for LPAR

- Existing programming languages do not support dynamic non-uniform data structures
- Fortran models support only uniform static arrays
  - Programmer manages bookkeeping
  - Some use C to do memory management – pointers problematic
- High-accuracy numerical methods employ irregular representations
  - Hierarchies of arrays
  - Particles
  - Non-uniform computational effort
- Important class of methods are non-uniform, locally structured
  - Particle methods
  - Multi-level and adaptive finite difference methods



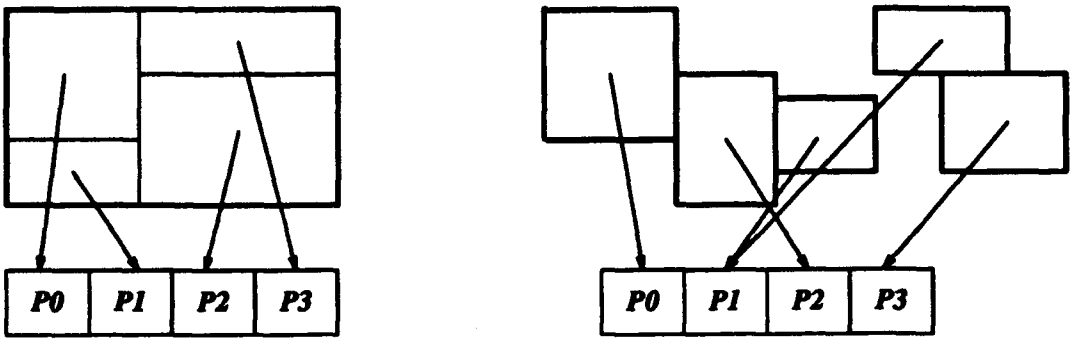


Overview of LPAR

- Coarse-grained data parallel programming model
- User is aware of parallelism, but at a high level
- Programmer has a clear model of how to achieve high performance but details are hidden
- Domain specific: above class of non-uniform, locally structured methods
- Portability across distributed memory MIMD architectures
  - Hides machine-dependent details
  - Uniform model of data structure manipulation
- Dynamic memory management facilities
  - Dynamic, irregular arrays
  - Coarse-grained irregular decompositions
  - Explicit management of program locality
- LPAR's dynamic array facilities may coexist with existing programming languages
  - LPAR provides coordination and control only
  - LPAR allows low-level performance tuning

LPAR Coding Example

- Applications consist of arrays individually assigned to processors



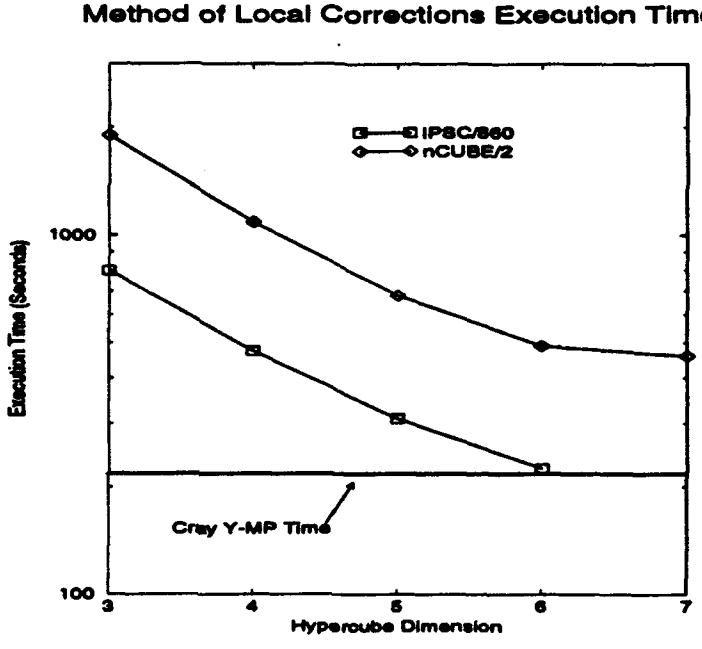
- Communication among arrays is managed by LPAR's high-level abstractions

```
function FillPatch( distributed (*) Map [*1] of Map [*2] of double bins,
                    Map [1:P] of Domain [*2] partition )

   distributed foreach I in dmn(bins)
      foreach J in dmn(bins)
         bins[I] <<= bins[J] on partition[J];
      end foreach
   end distributed foreach

end function
```

- Equivalent code is thousands of lines long in message passing Fortran or C

LPAR Implementation

- Prototype LPAR implemented as a C++ class library
  - Workstation for code development and debugging
  - Parallel architectures: iPSC/860 and nCUBE/2
- LPAR applications
  - Two dimensional fast N-body solver for vortex dynamics (shown in figure below as "Method of Local Corrections")
  - Two and three dimensional multigrid
  - Two and three dimensional conjugate gradient solvers
- Performance on 64 nodes of an iPSC/860 comparable to one processor of a Cray Y-MP



Current Status and Future Directions

- Software distribution in Fall of 1993
  - Anonymous FTP
  - NSF Metacenters
- Broaden architectural base
  - CM-5
  - Paragon
  - Networks of Workstations (PVM)
  - KSR-1
  - Cray C-90
- Compiler support
  - Automated load balancing
  - Performance enhancements
- New applications
  - Fast adaptive elliptic solvers for first principles simulations
  - Adaptive mesh refinement for hyperbolic conservations laws
  - Smoothed particle hydrodynamics
  - Genetic algorithms

PROCEEDINGS OF THE SIXTH SIAM CONFERENCE ON

PARALLEL PROCESSING FOR SCIENTIFIC COMPUTING

Volume I

Edited by Richard F. Sincovec
David E. Keyes
Michael R. Leuze
Linda R. Petzold
Daniel A. Reed

siam

PREFACE

The Sixth SIAM Conference on Parallel Processing for Scientific Computing, held in Norfolk, Virginia, on March 22-24, 1993, focused on themes from the High Performance Computing and Communications (HPCC) program and, in particular, on grand challenge problems. There has been a steady stream of accomplishments over the past few years on various grand challenge problems: major new scalable parallel computers and distributed networked computing have become available, new algorithms and computational methodologies have emerged, and software tools emphasizing generic multi-architecture capability are being perfected. In addition, a large number of scientists and engineers are adopting these new technologies. This conference provided a forum for these recent advances and promoted interdisciplinary collaborations among algorithm developers, numerical analysts, computer architects, and scientists and engineers. This conference, sponsored by the SIAM Activity Group on Supercomputing, was organized by Richard Sincovec, Chair, Oak Ridge National Laboratory; David Keyes, Yale University; Mike Leuze, Oak Ridge National Laboratory; Linda Petzold, University of Minnesota; and Dan Reed, University of Illinois. SIAM conducted this conference with the partial support of the National Science Foundation and the Department of Energy.

The proceedings contains papers from the invited talks, contributed talks of 30 minutes and 20 minutes, minisymposia, and poster presentations. The proceedings consists of two parts based on the conference themes: grand challenge problems or applications and support for grand challenge problems or infrastructure. Applications include computational fluid dynamics, geophysical modeling, materials science, molecular dynamics, and others. The infrastructure not only emphasizes the development and analysis of numerical algorithms for parallel computing as one would expect at a SIAM conference, but also includes parallel environments and tools.

Papers in this proceedings highlight improvements in algorithm design and software technology that are essential to achieving sustained high levels of computing system performance. Other papers illustrate improvements in algorithms and software and their successful application to important problems. Finally, papers on parallel and distributed systems explore new opportunities and challenges for algorithms, software, and applications. In totality, these papers represent a notable step in advancing high performance computing technology and facilitating its use on a wide range of problems in science, engineering, health care, education, national security, and other areas.

Researchers, scholars, students, educators, scientists, engineers, and managers who are using parallel computing, designing and analyzing parallel computers, or developing algorithms, tools and environments for parallel computers should find these proceedings of interest.

Richard F. Sincovec
Oak Ridge National Laboratory

Chapter 14
Section 1
Parallel Programming Models

An Implementation of the LPAR Parallel Programming Model for Scientific Computations*

Scott R. Kohn† Scott B. Baden‡

Abstract

LPAR is a portable coarse-grain parallel programming model for non-uniform structured scientific applications running on MIMD message passing architectures. Non-uniform applications, which include N-body methods and adaptive multilevel mesh methods, rely on complex dynamic data structures and are particularly difficult to implement on parallel computers. This paper introduces the LPAR programming abstractions and discusses some important implementation issues. We also present performance results on the Intel iPSC/860 and nCUBE/2 for a vortex dynamics application developed using LPAR.

1 Introduction

Recent developments in numerical methods for solving partial differential equations have emphasized elaborate, dynamic, non-uniform data structures. These methods attempt to place computational effort and accuracy in regions of high error or rapidly changing solutions. They are particularly attractive for solving local, non-uniform, time-dependent problems. Typical applications include fast N-body methods in molecular dynamics, astrophysics, and computational fluid dynamics; adaptive multilevel mesh methods such as multigrid, FAC and AFAC [13], and AMR [5]; and finite element methods.

An important characteristic of many of these numerical methods is that most communication is localized. For example, in fast N-body methods, force calculations are dominated by short-range interactions; grids in adaptive mesh methods only communicate with overlapping or adjacent grids. While these methods may require non-local communication and computation, such as the calculation of far-field forces, this work is typically inexpensive compared to the localized computation. A programming model which exploits this inherent locality can achieve better performance on parallel architectures by reducing communication overheads. Exploiting locality can also reduce memory requirements [2].

*This work was supported by NSF contract number ASC-9110793. Computer time on the iPSC/860 and nCUBE/2 at the San Diego Supercomputer Center was provided by a UCSD Division of Engineering Block Grant.

†Graduate Student, University of California – San Diego, Department of Computer Science and Engineering, 9500 Gilman Drive, La Jolla, CA 92093-0114. (skohn@cs.ucsd.edu).

‡Assistant Professor, University of California – San Diego (baden@cs.ucsd.edu).

Implementing dynamic non-uniform calculations on a message passing parallel architecture is a difficult task. Without support from a high-level parallel programming language, the programmer must partition data structures across distributed memory, balance workloads, and coordinate the communication of data values between processors. Important advances have been made in fine-grain programming models such as Fortran D [10], CM Fortran, and the proposed HPF. However, fine-grain models are inappropriate for certain classes of applications as they may incur high overhead costs in managing communication.

For example, Clark et al. [7] describe an implementation of the GROMOS molecular dynamics code in Fortran D. In one stage of the algorithm, the GROMOS code approximates nonbonded forces by calculating interactions only for atom pairs lying within a specified cutoff radius. In the Fortran D implementation, atoms are identified with unique integers used to access one dimensional arrays of position and velocity. However, this form of data representation destroys the physical locality of the original problem, since the arrays are distributed across processors according to array indices, which have little to do with the physical positions of the atoms. Fortran D's irregular array decompositions can be used to map nearby atoms to the same processor, restoring locality and reducing communication. However, because this mapping is unstructured and fine-grained, the application may incur substantial run-time overhead in analyzing irregular, unstructured communication patterns.

The LPAR abstractions, described in the following section, preserve the spatial locality inherent in many scientific applications and therefore avoid the potentially high overheads of fine-grain models.

2 The LPAR Programming Model Abstractions

LPAR [3] is a coarse-grain programming model for managing complex, dynamic calculations on MIMD message passing architectures. It provides high-level abstractions which exploit the locality found in non-uniform structured scientific computations. LPAR frees the programmer from dealing with low-level details involved in partitioning, synchronization, and communication. LPAR does not hide parallelism from the programmer; rather, parallelism is considered a fundamental component of program design. However, LPAR does provide powerful tools for managing the complexity of parallel code. Applications written in LPAR are portable across a wide range of MIMD architectures at performance comparable to "hand-coding."

LPAR is based in part on the scientific programming language FIDIL [11] and is designed to be embedded within a standard programming language such as C++ or HPF. LPAR adds two new abstract data types, the *map* and the *domain*. Maps are dynamically defined arrays with arbitrary bounds. The set of points which may be used to index a map is called the domain of the map. The domain is a first-class object in LPAR and is a generalization of array sections found in HPF. Figure 1a shows a two dimensional map `A` with a lower bound at point (1,3) and upper bound at (5,6); the domain of `A`, denoted by `dmn(A)`, is [1:5, 3:6].

Map types are not necessarily limited to floating point numbers or integers. LPAR allows maps of structures, maps of maps, or maps of particle lists. For example, in addition to representing a finite difference grid, a map may also be used to implement the chaining mesh [12] common in particle codes such as the vortex dynamics application described in Section 4.

Maps and domains are manipulated using LPAR's *domain calculus*. Maps and domains are embedded in a logically shared global coordinate system. Even though maps may logically overlap within this coordinate system, they do not physically share map elements.

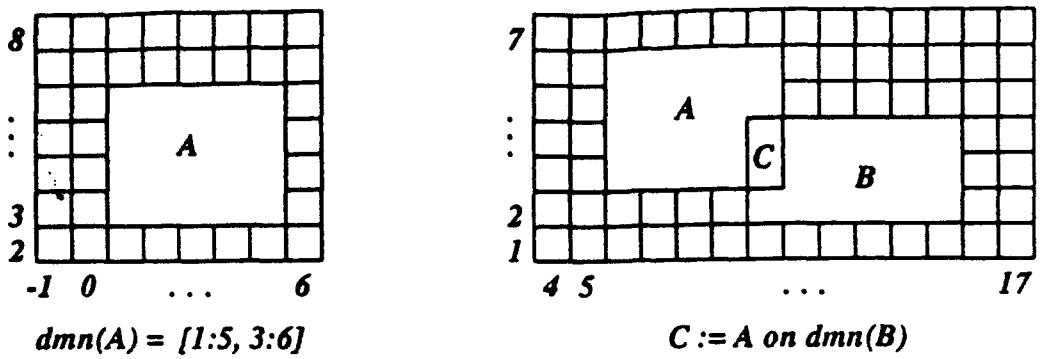


FIG. 1. *LPAR Maps: (a) Map and Domain (b) Map Restriction*

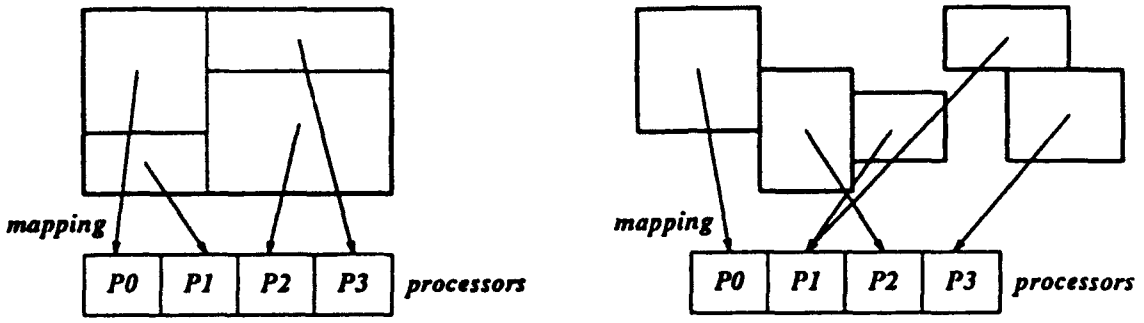


FIG. 2. *Decompositions: (a) Recursive Bisection (b) Refinement Structure*

For example, Figure 1b shows maps A and B and an underlying global coordinate system. Map C is assigned the portion of map A which overlaps the domain of B in the global coordinate system. This operation, called restriction, is written in the domain calculus as C := A on dmn(B). Note that even though maps A, B, and C logically overlap on [10, 3:4], the three maps are independent and may contain different data values in this region. In addition to restriction, the domain calculus defines operations for union, intersection, set difference, coordinate shift, and assignment.

Applications are parallelized in LPAR by decomposing the problem domain into a collection of maps which are then individually assigned to processors. This process is a coarse-grain analogue of Fortran D's irregular array decompositions. Algorithms for decomposing problem domains and assigning maps to processors are application-specific and are implemented in applications libraries. LPAR provides a common framework for specifying the decompositions and mappings, and the programmer is free to select the strategy which best matches the application. (Refer to [3] for a more detailed discussion.)

Collections of maps represent the high-level structure of the parallel computation. Because collections are coarse-grained decompositions, they typically contain $O(p)$ maps when running on p processors. Figure 2 shows two common decompositions and mappings. Figure 2a illustrates the decomposition of a rectangular region into four non-uniform maps using recursive bisection [4]. The collection of maps shown in Figure 2b is typical of a single level refinement structure for adaptive mesh methods.

Data dependencies among decomposed LPAR maps such as those in Figure 2 can be easily expressed using LPAR's domain calculus. These primitives allow the programmer to specify communication at a very high level; bookkeeping and synchronization details are completely hidden from the programmer. For example, consider the data dependencies shown in Figure 3, which typically arise in adaptive mesh refinement algorithms. Grids are usually extended to include a boundary region, also known as a ghost cell region or shadow region. These boundary regions are injected with data from the interior regions of

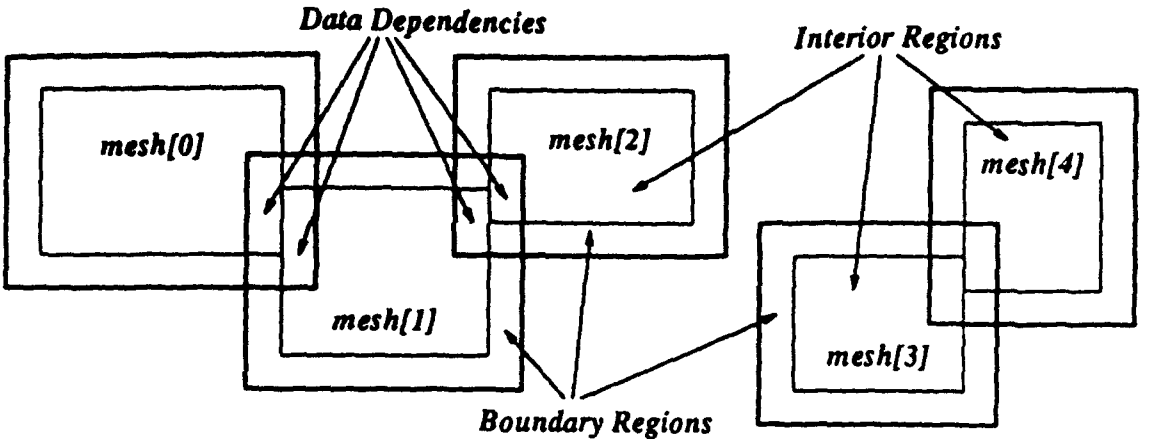


FIG. 3. *Data Dependencies (Shaded) for an Adaptive Mesh Refinement Algorithm*

```
forall i in {0, ..., 4}
     boundary := boundaryOf(mesh[i]);
     for j in {0, ..., 4}
          if (i ≠ j)
               interior := interiorOf(mesh[j]);
               dependence := boundary ∩ interior;
               mesh[i] on dependence := mesh[j] on dependence;
          end if
     end for
end forall
```

FIG. 4. *LPAR Pseudo-Code for Satisfying Data Dependencies*

logically overlapping grids. It is trivial to represent these data dependencies using LPAR's domain calculus. The shaded dependence regions can be expressed as the intersection of the boundary domain of one map and the interior domain of another map. LPAR pseudo-code which expresses the communication dependencies between maps is shown in Figure 4[1]. All communication, gathering and scattering of data values, and synchronization is managed within LPAR and is invisible to the programmer. Similar code written without LPAR would be several thousand lines long.

3 Implementation Issues

The primary LPAR implementation issue is how to avoid unnecessary communication between processors in satisfying data dependencies specified using the domain calculus. As shown in Figure 3, data dependency regions are typically a small fraction of the total map size. Furthermore, most interactions between maps are empty. LPAR is able to reduce communication overheads and improve performance by communicating only necessary data values and eliminating messages for spurious dependencies.

LPAR avoids unnecessary communication through run-time optimizations known as *live transmission analysis* [14]. Live transmission analysis determines if communication between processors is necessary and, if so, calculates the minimum set of data needed to satisfy the data dependencies. Analysis must be performed at run-time since map structures are not usually known at compile time. Adaptive algorithms may dynamically create and destroy maps. and data dependencies may change in time as workloads are rebalanced.

[1]For a detailed description of LPAR usage, refer to [3].

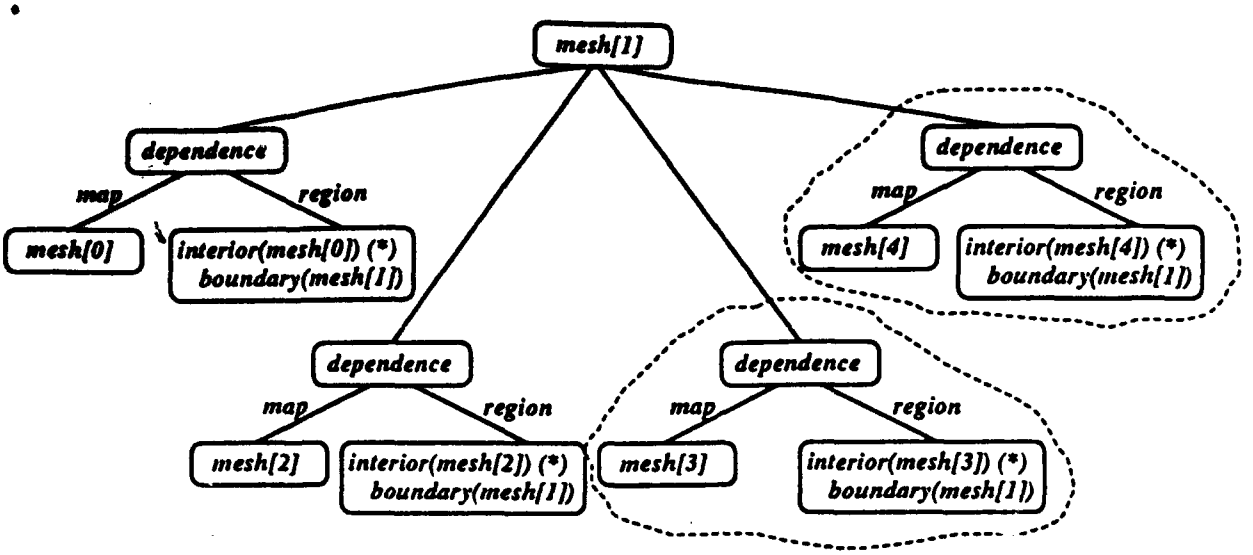


FIG. 5. *Live Transmission Analysis Data Dependence Tree for Map* `mesh[1]`

At run-time, LPAR builds and evaluates data dependence trees which express data dependencies between maps. For example, after executing the code shown in Figure 4 for the adaptive mesh problem (Figure 3), the LPAR implementation would have built the data dependence tree in Figure 5 for map `mesh[1]`. This tree describes the four possible data dependencies for `mesh[1]`. Each **dependence** node includes information about the map object containing the needed data and the region of the dependence.

Evaluation and optimization of data dependence trees is demand-driven. The evaluation and optimization pass on the data dependence tree determines the minimum set of data to be obtained from other processors and prunes portions of the tree in which domain intersections are empty. In Figure 5, the intersections in the two rightmost branches of the tree, which are outlined, are empty; thus, LPAR would prune these branches of the tree. After completing the optimization pass on the data dependence tree, LPAR sends data requests to other processors for each remaining dependence. After receiving these requests, processors reply with their map values. Although two messages are communicated for every data dependence – one request message and one response message – this overhead does not significantly impact performance and could be eliminated by an optimizing compiler.

After all data requests have been satisfied, LPAR discards its communication schedules. LPAR could amortize the cost of creating communication schedules by saving and reusing them later in the computation. PARTI [9] takes this approach. However, LPAR applications are dynamic and their communication schedules change frequently, providing few opportunities for reuse. Fortunately, collections typically contain relatively few maps since they represent coarse-grained decompositions. Therefore, LPAR communication schedules are many times smaller than the schedules produced by libraries such as PARTI, which use fine-grained decompositions.

When evaluating the domains of dependence, LPAR needs to know the domains of all maps, even those which lie on other processors. For example, although `mesh[2]` is only assigned to one processor, all processors must have a copy of the domain information of `mesh[2]` when performing tree optimizations. Therefore, domain descriptions, but not map elements, are replicated on all processors. The memory overhead of this replication is small, since maps have a very simple structure and thus can be represented in a compact format. Furthermore, the cost of maintaining consistency of this distributed domain information is

minimal. Although the domains of maps may change in the course of a calculation, they typically change at approximately the same time in a repartitioning or regridding section of the code. Therefore, LPAR is able to globally update domain information in unison.

Although live transmission analysis is performed at run-time, because the maps are assumed to have a simple structure and the domain information is replicated across the processors, the overhead of these optimizations is negligible. This overhead is discussed in more detail in the following section.

4 A Fast N-Body Application

A prototype implementation of LPAR has been written as a C++ class library. LPAR is currently running on a Sun workstation, the Intel iPSC/860, and the nCUBE/2. Both the iPSC/860 and nCUBE/2 are MIMD message passing architectures. In the near future, LPAR will also be ported to the Intel Paragon and the CM-5.

LPAR was used to parallelize a two dimensional vortex dynamics calculation which solves the vortex-stream formulation of the Euler equations. The vortex dynamics algorithm discretizes vorticity onto marker vortices which are advected according to a velocity field. Vortex blob velocities are calculated using the Method of Local Corrections [1], in which computation is dominated by the calculation of many body forces. This method divides the velocity evaluation into two components: far-field evaluations and local corrections. In the far-field phase, vortex velocities are projected onto a grid and Poisson's equation is solved to obtain a discrete global velocity field. This velocity field is locally corrected in the local corrections step, which directly calculates particle-particle interactions to correct the velocity contributions of nearby vortices. This application is typical of a broader class of fast N-body methods.

To maintain the spatial relationships among particles, they are sorted into a two dimensional array of bins. The LPAR code decomposes this binning array into a collection of maps using recursive bisection. Because the particles move as the calculation progresses, the binning array is rebalanced every time step. Communication of particles between maps is managed using LPAR's domain calculus.

The LPAR application consists of approximately 3,000 lines of C++ code and was compiled using the GNU g++ compiler. Figure 6 compares the LPAR code to a "hand-coded" implementation in Fortran 77 (13,000 lines) for a 50 time step run with 12,848 vortices. LPAR results are presented for the iPSC/860 and nCUBE/2; the Fortran version ran on the iPSC/860 and one processor of a Cray Y-MP8/64.

Figure 7 separates LPAR execution times on the iPSC/860 and nCUBE/2 into four categories: *local work*, *global work*, *communication*, and *rebalancing*. *Local work* measures localized computation and *global work* reflects time spent solving Poisson's equation; both sections contain immeasurably small LPAR overheads consisting of an occasional barrier synchronization. Most LPAR overhead is found in the *communication* and *rebalancing* portions of the code. In *communication*, LPAR was used to obtain vortices from maps on other processors and to reassign vortices that had advected off of the local partition. *Rebalancing* used LPAR's domain calculus to reassign particles to processors while rebalancing workloads.

Table 1 compares the LPAR and "hand-coded" communication times. The LPAR overhead in this table is approximately three times greater than the overhead in the Fortran application. Some of this overhead can be attributed to the more general and high-level gathering and scattering facilities provided in the LPAR version of the code.

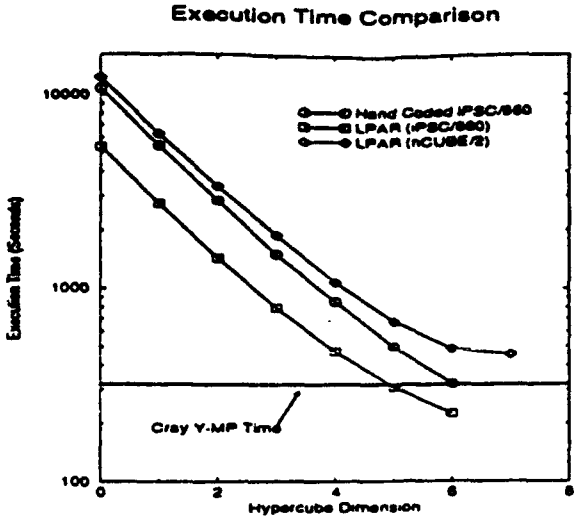


FIG. 6. *LPAR and Hand-Coded Execution Times for a Vortex Dynamics Calculation*

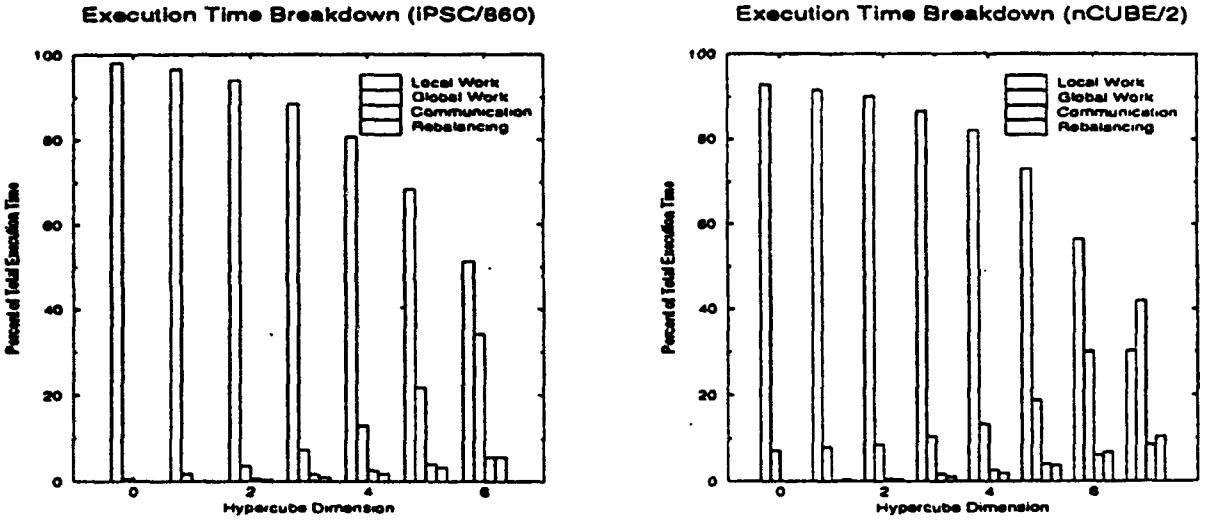


FIG. 7. *LPAR Execution Time Breakdown on the iPSC/860 and the nCUBE/2*

LPAR's request-reply communications protocol accounts for the remainder of the difference. However, as shown in Figure 7, the cumulative LPAR overhead still represents a small fraction of the total execution time. The ease of programming using LPAR more than justifies the small increase in overhead.

5 Conclusions

LPAR is a new coarse-grain, portable programming model for MIMD message passing architectures. LPAR provides high-level abstractions which free the programmer from low-level details involved in partitioning, synchronization, and communication. LPAR has been used to implement a non-uniform vortex dynamics code which is typical of many N-body methods. Performance results on the iPSC/860 and the nCUBE/2 show that LPAR can be used to parallelize an application without sacrificing performance. Collaboration is underway to develop an LPAR application for the first principles simulation of real materials. This work will allow theoretical material scientists to employ more ambitious simulation techniques, broadening the range of materials which can be studied. LPAR

TABLE 1
Comparison of Communication Overheads on the iPSC/860

Processors		1	2	4	8	16	32	64
Communication (seconds)	Hand Coded	0	1.07	3.30	3.55	4.15	3.38	3.30
	LPAR	0	3.19	8.93	12.9	12.5	12.3	12.6

will also be used to develop an adaptive mesh refinement code for hyperbolic PDE's in computational fluid dynamics. We plan to apply the LPAR abstractions to the irregular but locally coupled meshes used in finite element methods [8] and to the irregularly coupled regular meshes [6] used in aerodynamics. We believe that LPAR will substantially reduce the development time of these codes without impacting performance.

References

[1] C. R. Anderson, *A method of local corrections for computing the velocity field due to a distribution of vortex blobs*, Journal of Computational Physics, 62 (1986), pp. 111–123.

[2] S. B. Baden and S. Kohn, *A comparison of load balancing strategies for particle methods running on mimd multiprocessors*, in Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing, Houston, Texas, March 25–27 1991. Also Technical Report CS91-199, Computer Science and Engineering, University of California, San Diego.

[3] ——, *Lattice parallelism: A parallel programming model for non-uniform, structured scientific computations*, Tech. Rep. CS92-261, University of California - San Diego, CSE 0114, 9500 Gilman Drive, La Jolla, CA 92092-0114, August 1992. Also available via anonymous `ftp` from `cs.ucsd.edu` in directory `pub/baden/lpar`.

[4] M. J. Berger and S. H. Bokhari, *A partitioning strategy for nonuniform problems on multiprocessors*, IEEE Transactions on Computers, C-36 (1987), pp. 570–580.

[5] M. J. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, Journal of Computational Physics, 82 (1989), pp. 64–84.

[6] C. Chase, K. Crowley, J. Saltz, and A. Reeves, *Parallelization of irregularly coupled regular meshes*, Tech. Rep. 92–1, ICASE, Hampton, VA, January 1992.

[7] T. W. Clark, R. V. Hanxleden, K. Kennedy, C. Koelbel, and L. R. Scott, *Evaluating parallel languages for molecular dynamics computations*, in Proceedings of the 1992 Scalable High Performance Computer Conference, Williamsburg, Virginia, March 1992, pp. 98–105.

[8] R. Das, D. Mavriplis, J. Saltz, S. Gupta, and R. Ponnusamy, *The design and implementation of a parallel unstructured euler solver using software primitives*, Tech. Rep. 92–12, ICASE, Hampton, VA, March 1992.

[9] R. Das, R. Ponnusamy, J. Saltz, and D. Mavriplis, *Distributed memory compiler methods for irregular problems — data copy reuse and runtime partitioning*, Tech. Rep. 91–73, ICASE, Hampton, VA, September 1991.

[10] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C.-W. Tseng, and M.-Y. Wu, *Fortran d language specification*, Tech. Rep. TR90-141, Deptartment of Computer Science, Rice University, Houston, TX, December 1989.

[11] P. N. Hilfinger and P. Colella, *Fidil: A language for scientific programming*, Tech. Rep. UCRL-98057, Lawrence Livermore National Laboratory, January 1988.

[12] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, McGraw-Hill, 1981.

[13] S. McCormick and D. Quinlan, *Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Performance results*, Parallel Computing, 12 (1989), pp. 145–156.

[14] L. Semenzato and P. Hilfinger, *Arrays in fidil*, in Proceedings of the First International Workshop on Arrays, Functional Programming, and Parallel Systems, July 1990.