Software and hardware requirements for some applications of parallel computing to industrial problems

Geoffrey C. Fox
*Syracuse University, Northeast Parallel Architectures Center*

Follow this and additional works at: [https://surface.syr.edu/npac](https://surface.syr.edu/npac)

*Part of the Computer Engineering Commons*

**Recommended Citation**


[https://surface.syr.edu/npac/38](https://surface.syr.edu/npac/38)

This Working Paper is brought to you for free and open access by the College of Engineering and Computer Science at SURFACE. It has been accepted for inclusion in Northeast Parallel Architecture Center by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.
Software and Hardware Requirements for Some Applications of Parallel Computing to Industrial Problems

Geoffrey C. Fox
gcf@npac.syr.edu
http://www.npac.syr.edu

Northeast Parallel Architectures Center
111 College Place
Syracuse University
Syracuse, New York 13244-4100

Abstract

We discuss the hardware and software requirements that appear relevant for a set of industrial applications of parallel computing. These are divided into 33 separate categories, and come from a recent survey of industry in New York State. The software discussions include data parallel languages, message passing, databases, and high-level integration systems. The analysis is based on a general classification of problem architectures originally developed for academic applications of parallel computing. Suitable hardware architectures are suggested for each application. The general discussion is crystalized with three case studies: computational chemistry, computational fluid dynamics, including manufacturing, and Monte Carlo Methods.

1 Introduction

This paper combines a recent survey of industrial applications with a problem classification developed for largely academic applications. We show how this allows one to isolate the parallel computing hardware and software characteristics needed for each problem. The industrial applications come from a survey undertaken of New York State industry in 1991 and 1992. Further details of the survey will be found in [Fox:92e], [Fox:94b], [Fox:94h], [Fox:94c], [Fox:94i], [Mills:93a]. Here, we summarize relevant features of it in Section 2. Section 3 reviews and extends a classification of problem architectures originally developed in 1988 from a rather complete survey of parallel applications at the time [Fox:88b], [Fox:88tt], [Fox:91g], [Fox:94a].
In Section 4, we show how the different problem categories or architectures are addressed by parallel software systems with different capabilities. We give illustrative examples, but not an exhaustive list of existing software systems with these characteristics. We consider High Performance Fortran and its extensions as a data parallel language; message passing systems, such as those supplied with commercial multicomputers; as well as approaches to software integration. In Section 3, we point that our old classification of problems omitted what we can call metaproblems—problems built up from several components—each of which could have its own parallelization issues.

In Section 5, we combine the previous three sections and describe informally the problem architecture, and possible software and hardware needs for a selection of industrial applications. We have grouped our discussion into three broad case studies: computational chemistry; computational fluid dynamics and manufacturing; Monte Carlo methods. Each case study covers several of the 33 industrial areas identified in Section 2. Further, in each case study, we find good examples of the different problem architectures and show that each application area requires a rich software and hardware architecture support.

2 Industrial Applications

In 1991–92, New York State funded a survey of the industrial opportunities for parallel computing. This was part of a project, ACTION (Advanced Computing Technology is an Innovative Opportunity Now), at Syracuse University. ACTION's goal was to accelerate the use of parallel computing into the industry of New York State. The purpose of the survey was to isolate some initial projects and, more generally, to provide information on which to develop a long-term plan for ACTION. Relevant information included the basic parallel computing software and algorithm technologies needed for particular industry sectors. In this paper, we concentrate on the software capabilities needed for these applications, and also the appropriate parallel machine architectures. Further discussion of the survey technique, particular companies interviewed, and the detailed nature of the applications will be found in [Fox:92e], [Fox:94a], and papers cited earlier. The survey has inevitable limitations. There are many important applications—oil exploration is a good example—that are not well represented in New York State. Further, the survey was not complete within its limited geographic region, and is still being extended.
Tables 1–4 summarize the industrial opportunities for parallel computing in the form we will use them. Some 50 different applications used in the survey have been broken up into 17 distinct areas. This is certainly somewhat arbitrary, and there are many overlaps (and omissions as discussed). The importance, difficulty of implementation, and degree of risk also differ from case to case. However, these issues will not be discussed here.
Table 1: Guidelines used in Developing Categories of Industrial and Government Applications of HPCC shown in Tables 3–4

- Define information generally to include both CNN headline news and the insights on QCD gotten from lattice gauge theories. There are four broad categories.

- **Information Production** (e.g., Simulation)
  - Major concentration of MPP and HPCC at present

- **Information Analysis** (e.g., Extraction of location of oil from seismic data, Extraction of customer preferences from purchase data)
  - Growing area of importance and Short term major MPP opportunity in decision support combined with parallel databases

- **Information Access and Dissemination—InfoVision** (e.g., Transaction Processing, Video-On-Demand)
  - Enabled by National Information Infrastructure
  - Very promising medium term market for MPP but need the NII to be reasonably pervasive before area “takes off”
  - MPPs used as high performance, high capacity, multi-media servers

- **Information Integration**
  - Integrates Information Production, Analysis and Access, e.g.,
    - Decision support in business
    - Command and Control for Military
    - Concurrent Engineering and Agile Manufacturing
  - Largest Long Term Market for MPP
Table 2: Abbreviations used in Tables 3–4 of Industrial Applications of HPCC

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive</td>
<td>Software for Irregular Loosely Synchronous Problems handled by pC++, HPF extensions, Message Passing (Table 6)</td>
</tr>
<tr>
<td>Asyncsoft</td>
<td>Parallel Software System for (particular) class of asynchronous problems (Table 6)</td>
</tr>
<tr>
<td>CFD</td>
<td>Computational Fluid Dynamics</td>
</tr>
<tr>
<td>ED</td>
<td>Event Driven Simulation</td>
</tr>
<tr>
<td>FD</td>
<td>Finite Difference Method</td>
</tr>
<tr>
<td>FEM</td>
<td>Finite Element Method</td>
</tr>
<tr>
<td>HPF</td>
<td>High Performance Fortran</td>
</tr>
<tr>
<td>HPF+</td>
<td>Natural Extensions of HPF</td>
</tr>
<tr>
<td>Integration</td>
<td>Software to integrate components of metaproblems (Table 6)</td>
</tr>
<tr>
<td>MPF</td>
<td>Fortran plus message passing for loosely synchronous programming support</td>
</tr>
<tr>
<td>PDE</td>
<td>Partial Differential Equation</td>
</tr>
<tr>
<td>TS</td>
<td>Time Stepped Simulation</td>
</tr>
<tr>
<td>VR</td>
<td>Virtual Reality</td>
</tr>
</tbody>
</table>

Note on Language: HPF, MPF use Fortran for illustration, one can use parallel C, C++ or any similar extensions of data parallel or message passing languages
Table 3: Five Categories of Problems

<table>
<thead>
<tr>
<th>Problem Architecture</th>
<th>Overall Software Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synchronous</strong>: Data Parallel</td>
<td>Data Parallel</td>
</tr>
<tr>
<td>Tightly coupled and software needs to exploit features of problem structure to get good performance. Comparatively easy as different data elements are essentially identical.</td>
<td></td>
</tr>
<tr>
<td><strong>Loosely Synchronous</strong>: Data Parallel</td>
<td>Data Parallel</td>
</tr>
<tr>
<td>As above, but data elements, and/or their linkage, are not identical. Still parallelizes due to macroscopic time synchronization.</td>
<td></td>
</tr>
<tr>
<td><strong>Asynchronous</strong>:</td>
<td>Difficult Task parallel</td>
</tr>
<tr>
<td>Functional (or data) parallelism that is irregular in space and time. Often loosely coupled and so need not worry about optimal decompositions to minimize communication. Hard to parallelize (massively) unless...</td>
<td></td>
</tr>
<tr>
<td><strong>Embarrassingly parallel</strong>:</td>
<td>Possible in most software</td>
</tr>
<tr>
<td>Essentially independent execution of disconnected components (can involve reductions).</td>
<td></td>
</tr>
<tr>
<td><strong>Metaproblems</strong></td>
<td>Coarse grain task parallel</td>
</tr>
<tr>
<td>Asynchronous collection of (loosely) synchronous components where these programs themselves can be parallelized</td>
<td></td>
</tr>
<tr>
<td>Item</td>
<td>Application Area and Examples</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>Computational Fluid Dynamics</td>
</tr>
<tr>
<td></td>
<td>• Aerospace</td>
</tr>
<tr>
<td></td>
<td>• Military, Civilian</td>
</tr>
<tr>
<td></td>
<td>Vehicles</td>
</tr>
<tr>
<td></td>
<td>• Propulsion</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Structural Dynamics</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Electromagnetic Simulation</td>
</tr>
<tr>
<td></td>
<td>• Antenna Design</td>
</tr>
<tr>
<td></td>
<td>• Stealth Vehicles</td>
</tr>
<tr>
<td></td>
<td>• Noise in high</td>
</tr>
<tr>
<td></td>
<td>frequency circuits</td>
</tr>
<tr>
<td></td>
<td>• Mobile Phones</td>
</tr>
<tr>
<td>4</td>
<td>Scheduling</td>
</tr>
<tr>
<td></td>
<td>• Manufacturing</td>
</tr>
<tr>
<td></td>
<td>• Transportation</td>
</tr>
<tr>
<td></td>
<td>(Dairy delivery to Military deployment)</td>
</tr>
<tr>
<td></td>
<td>• University Classes</td>
</tr>
<tr>
<td></td>
<td>• Airline Scheduling</td>
</tr>
<tr>
<td></td>
<td>of crews, planes</td>
</tr>
<tr>
<td></td>
<td>in static or dynamic</td>
</tr>
<tr>
<td></td>
<td>(Syracuse snow storm) cases</td>
</tr>
<tr>
<td>5</td>
<td>Environmental Modeling —</td>
</tr>
<tr>
<td></td>
<td>Earth/Ocean/Atmospheric</td>
</tr>
<tr>
<td></td>
<td>Simulation</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4: Industrial HPCC Applications 6 to 10: SIMULATION

<table>
<thead>
<tr>
<th>Item</th>
<th>Application Area and Examples</th>
<th>Problem Comments</th>
<th>Machine and Software</th>
</tr>
</thead>
</table>
| 6    | Environmental Phenomenology   | • Empirical Models
—Complex Systems,  | • Some SIMD but
[Lead Concentration | • MIMD more
in blood]             | natural         |
                  | • Monte Carlo and           | • HPF             |
                  |                  Histograms          |                   |
| 7    | Basic Chemistry               | • Calculate Matrix Elements | Probably SIMD with perhaps
• Chemical Potentials | • Matrix Eigenvalue | SIMD possible
• Elemental Reaction | determination, Inversion,  |
Dynamics          | Multiplication              | • HPF             |
| 8    | Molecular Dynamics            | • Particle Dynamics  | HPF(+) except
in Physics & Chemistry | with irregular cutoff forces    | need MPF for
• Biochemistry     | • Fast Multipole Methods     | fast multipole
• Discrete Simulation | • Mix of PDE and           |
Monte Carlo for   | Particle methods in PIC and  |                   |
CFD (DSMC)        | DSMC                       |                   |
• Particle in the Cell (PIC) | | |
| 9    | Economic Modelling            | Single financial instrument by | SIMD, HPF |
• Real Time Optimization | Monte Carlo                | MIMD or SIMD      |
• Mortgaged backed Securities | | with Integration Software |
• Option Pricing   | Full Simulations of complete portfolios | |
| 10   | Network Simulations           | • Sparse matrices     | MIMD               |
• Electrical Circuit | • Zero structure defined by |
• Microwave and VLSI | connectivity              | HPF for matrix |
• Biological (neural) Circuit | | elements |
|      |                               |                  | MPF or library |
|      |                               |                  | for matrix solve |
Table 4: Industrial HPCC Applications 11 to 13: SIMULATION

<table>
<thead>
<tr>
<th>Item</th>
<th>Application Area and Examples</th>
<th>Problem Comments</th>
<th>Machine and Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Particle Transport Problems</td>
<td>Monte Carlo Methods as in neutron transport for (nuclear) explosion simulations</td>
<td>MIMD HPF</td>
</tr>
<tr>
<td>12</td>
<td>Graphics (rendering)</td>
<td>Several Operational Parallel Ray tracers Distributed model hard</td>
<td>HPF for simple ray tracing but MPF for best algorithms MIMD &amp; Asyncsoft for distributed database</td>
</tr>
<tr>
<td>13</td>
<td>Integrated Complex System Simulations</td>
<td>Event Driven (ED) and Time stepped (TS) simulations Virtual Reality Interfaces Database backends Interactive</td>
<td>Timewarp or other Event Driven Simulation needs Appropriate Asyncsoft Integration Software Database HPF+ for TS Simulation</td>
</tr>
</tbody>
</table>

- Hollywood
- Virtual Reality

- Defense (SIMNET, Flight Simulators)
- Education (SIMCITY)
- Multimedia/VR in Entertainment
- Multiuser Virtual Worlds
- Chemical and Nuclear Plants
<table>
<thead>
<tr>
<th>Item</th>
<th>Application Area and Examples</th>
<th>Problem Comments</th>
<th>Machine and Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Seismic and Environmental Data Analysis</td>
<td>• Parallel Computers already important but&lt;br&gt;• No oil in New York State</td>
<td>• SIMD useful but&lt;br&gt;• MIMD might be necessary&lt;br&gt;• HPF</td>
</tr>
<tr>
<td>15</td>
<td>Image Processing&lt;br&gt;• Medical Instruments&lt;br&gt;• EOS (mission to Planet Earth)&lt;br&gt;• Defense Surveillance&lt;br&gt;• Computer Vision</td>
<td>• Many commercial Applications of Defense Technology&lt;br&gt;• Component of many “metaproblems” (Information Integration category)&lt;br&gt;• e.g., Computer Vision in Robotics</td>
<td>Metacomputer&lt;br&gt;Low Level Vision is&lt;br&gt;• SIMD and HPF&lt;br&gt;Medium/High Level Vision is&lt;br&gt;• MIMD and HPF(+) Software Integration needs Aynsoft and Database</td>
</tr>
<tr>
<td>16</td>
<td>Statistical Analysis Packages and Libraries</td>
<td>• Optimization&lt;br&gt;• Histograms&lt;br&gt;• See application area 4</td>
<td>HPF + and especially C++ analogues is excellent for many libraries</td>
</tr>
<tr>
<td>17</td>
<td>Healthcare Fraud&lt;br&gt;• Inefficiencies&lt;br&gt;• Securities Fraud&lt;br&gt;• Credit Card Fraud</td>
<td>Linkage Analysis of Data records for correlations and outlier detection</td>
<td>• SIMD or MIMD&lt;br&gt;• Parallel Relational Database access&lt;br&gt;plus application area 16</td>
</tr>
<tr>
<td>18</td>
<td>Market Segmentation&lt;br&gt;• Mail Order&lt;br&gt;• Retail&lt;br&gt;• Banking</td>
<td>Sort and classify records to determine customer preference by region from city to even individual home</td>
<td>• Some cases are SIMD&lt;br&gt;• Parallel Database&lt;br&gt;plus application area 16</td>
</tr>
</tbody>
</table>
### Table 4: Industrial HPCC Applications 19 to 22 for Information Access

**InfoVision — Information, Video, Imagery and Simulation on Demand**

<table>
<thead>
<tr>
<th>Item</th>
<th>Application Area</th>
<th>Comments</th>
<th>Problem Structure</th>
<th>Machine &amp; Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td><strong>Transaction Processing</strong>&lt;br&gt;• ATM (automatic teller machine)</td>
<td>Database-most transactions short. As add “value” this becomes Information Integration</td>
<td>Emarrassingly Parallel</td>
<td>MIMD Software</td>
</tr>
<tr>
<td>20</td>
<td><strong>Collaboration</strong>&lt;br&gt;• Telemedicine&lt;br&gt;• Collaboratory for Research&lt;br&gt;• Education&lt;br&gt;• Business</td>
<td>Research Center or doctor(s)—patient interaction without regard to physical location</td>
<td>Asynchronous</td>
<td>High Speed Network</td>
</tr>
<tr>
<td>21</td>
<td><strong>Text on Demand</strong>&lt;br&gt;• Digital (existing) libraries&lt;br&gt;• ERIC Education database,&lt;br&gt;• United Nations-Worldwide newspapers</td>
<td>Multimedia database (see areas 22, 23)&lt;br&gt;Full text search</td>
<td>Emarrassingly Parallel</td>
<td>MIMD Database</td>
</tr>
<tr>
<td>22</td>
<td><strong>Video on Demand</strong>&lt;br&gt;• Movies, News (CNN NewsSource &amp; Newsroom),&lt;br&gt;• Current cable,&lt;br&gt;• United Nations-Policy Support</td>
<td>Multimedia Database&lt;br&gt;Interactive VCR, Video Browsing, Link of video and text database</td>
<td>Emarrassingly Parallel for multiple Users&lt;br&gt;Interesting parallel compression</td>
<td>MIMD Database&lt;br&gt;Video Editing Software&lt;br&gt;SIMD compression</td>
</tr>
</tbody>
</table>
Table 4: Industrial HPCC Applications 23 to 24 for Information Access InfoVision—Information, Video, Imagery and Simulation on Demand

<table>
<thead>
<tr>
<th>Item</th>
<th>Application Area</th>
<th>Comments</th>
<th>Problem Structure</th>
<th>Machine &amp; Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Imagery on Demand</td>
<td>Multimedia database, Image Understanding for Content searching and (terrain) medical feature identification</td>
<td>Metaproblem Embarrassingly Parallel plus Loosely Synchronous Image Understanding</td>
<td>MIMD but much SIMD image analysis</td>
</tr>
<tr>
<td>24</td>
<td>Simulation on Demand</td>
<td>Multimedia map database, Generalized flight simulator, Geographical Information System</td>
<td>Synchronous terrain rendering with Asynchronous Hypermedia</td>
<td>SIMD terrain engine (parallel rendering) SIMD database <strong>Integration</strong> software</td>
</tr>
</tbody>
</table>
These involve combinations of Information Production, analysis, Access and Dissemination and thus need the Integration of the various Software and Machines Architecture Issues discussed under previous application areas.

Sometimes Called System of Systems

**25: Military and Civilian Command and Control (C², C³, C⁴...)**
- Battle Management, Command, Control, Communication, Intelligence and Surveillance (BMC³IS)
- Military Decision Support
- Crisis Management—Police and other Government Operations
- SIMNET simulates this and with people and computers in the loop has many of same issues

**26 to 28: Applications of InfoVision Services**
- Generalize CompuServe, Prodigy, America Online, Dialog and Other Information Services
- **26: Decision Support for Society**
  - Community Information Systems
  - Travel and Generalized Yellow Page Services
- **27: Business Decision Support**—One example is:
  - Health Care with Image and Video databases supporting telemedicine
- **28: Public Administration and Political Decision Support**
  - Government Information Systems
  - Maxwell School at Syracuse University teaches use of realtime video to aid world wide decisions (United Nations)
Table 4: Information Integration Applications 29 to 33

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>29:</td>
<td>Real-Time Control Systems</td>
</tr>
<tr>
<td></td>
<td>- Robotics uses Imagery to make decisions (control vehicles)</td>
</tr>
<tr>
<td></td>
<td>- Energy management controls power use and generation</td>
</tr>
<tr>
<td>30:</td>
<td>Electronic Banking</td>
</tr>
<tr>
<td></td>
<td>- Requires Security, Privacy, Electronic Cash, etc.</td>
</tr>
<tr>
<td>31:</td>
<td>Electronic Shopping</td>
</tr>
<tr>
<td>32:</td>
<td>Agile Manufacturing—Multidisciplinary Design and Concurrent Engineering</td>
</tr>
<tr>
<td></td>
<td>- Combines CAD with Applications 1 to 3</td>
</tr>
<tr>
<td></td>
<td>- Requires major changes to Manufacturing Infrastructure and Approach</td>
</tr>
<tr>
<td>33:</td>
<td>Education</td>
</tr>
<tr>
<td></td>
<td>- InfoMall Living Textbook—6 Schools on ATM network linked to HPCC InfoVision Servers at NPAC [Mills:95a]</td>
</tr>
</tbody>
</table>

Table 1 describes the general guidelines used in organizing Table 4. Note that we did not directly cover academic areas, and a more complete list (which included our industrial table) was produced by the Petaflops meeting [Peta:94a]. Notice that Tables 1–4 are organized around the concept of “information.” This corresponded to an early realization from the survey that the major industrial opportunities for HPCC in New York State were information related. Thus, for instance, simulation is subtitled “Information Production” with say, computational fluid dynamics simulations providing information to be used in either manufacturing (application 32) or education (application 33). It is not directly relevant to this paper, but the results of this survey caused the ACTION program to refocus its efforts and evolve into InfoMall [Fox:93c], [Fox:94f], [Fox:94h], [Fox:95b], [Infour:95a], [Mills:94a]. Here, “Info” refers to the information based application focus and “Mall” to the use of a virtual corporation (groups of “shareholders”) to produce the complex integrated applications enabled by HPCC.

The first column of Table 4 contains the area label and some sample applications. Algorithmic and other comments are in column two. The third and fourth columns describe, respectively, the problem architecture.
and an estimate of appropriate parallel software approach. The background for these two columns is described in the following two sections.

This paper is not intended to advocate a particular parallel software environment or language. Rather, we want to describe the broad capabilities in the parallel programming paradigm needed for the applications of Table 4. We believe that the programming functionality needed by a particular application is broadly determined by the problem architecture described in the following section. In discussing software needs, we do not discuss all the components of the parallel software environment and just those relevant for expressing problems.

For this reason, we use broad software classifications using, for instance, MPF (Fortran plus message passing) as typical of all similar explicit messaging systems— one could substitute here C plus message passing, or Fortran M programming environments. Again, PVM, MPI, or any such message passing system could be used without changing the significance of the tables. High Performance Fortran is used as a typical data parallel language, although this has an evolving definition and similar C++ environments could well be more attractive, and can be substituted in the table.

3 Problem Architectures

We have described our classification of problem architectures several times before, but here we just summarize it.

This classification [Angus:90a], [Denning:90a], [Fox:88b;90p;91g;94a], was deduced from our experience at Caltech combined with a literature survey that was reasonably complete up to the middle of 1989. At Caltech, we developed some 50 applications on parallel machines, 25 of which led to publications in the scientific literature, describing the results of simulations performed on our parallel computers [Fox:87d] [Fox:88a], [Fox:88e], [Fox:89a]. Our Caltech work was mainly on the hypercube, but the total of 300 references used in original classification covered work on the Butterfly, transputers, the SIMD Connection Machine, and DAP. We originally identified three temporal structures and one especially important (as it was so simple) spatial structure, which are the first four entries in Table 4. Chapter 3 of [Fox:94a] describes a “complex systems” approach to computation and introduces the spatial and temporal structure of problems and computers. We studied software as a mapping (Figure 1) of problems to computers with the software structure determined by the structure (architecture) of
both the individual complex systems—computers and problems—and their interrelation. In Figure 2, we summarize issues in the spatial-temporal plane. “Space” here refers to data (problem) or nodes and their linkage (computer). “Time” is iteration number and simulation time (problem) or counts clock cycles (computer).

The three general temporal structures are called synchronous, loosely synchronous, and asynchronous. The temporal structure of a problem is analogous to the hardware classification into SIMD and MIMD. Further detail is contained in the spatial structure or computational graph of Figure 3a describing the problem at a given instant of simulation time [Fox:88tt]. This is important in determining the performance, as shown in Chapter 3 of [Fox:94a] of an implementation, but it does not affect the broad software issues discussed here. In Table 4, we only single out one special spatial structure, “embarrassingly parallel,” where there is little or no connection between the individual parallel program components. For embarrassingly parallel problems, illustrated in Figure 4, the synchronization (both software and hardware) issues are greatly simplified.

Synchronous problems are data parallel in the language of Hillis [Hillis:87a] with the restriction that the time dependence of each data point is governed by the same algorithm. Both algorithmically and in the natural SIMD im-
Figure 2: Issues Affecting Relation of Machine, Problem, and Software Architecture

Figure 3: (a) Synchronous, Loosely Synchronous (Static), and (b) Asynchronous (Dynamic) Complex Systems with their Space-Time Structure
Essentially Independent Parallel Processes

Example: Divide large database among processors and independently search each portion of database to answer query.

Figure 4: Embarrassingly Parallel Problem Class

Implementation, the problem is synchronized microscopically at each computer clock cycle. Such problems are particularly common in academic applications as they naturally arise in any description of some world in terms of identical fundamental units. This is illustrated in Figure 5 by quantum chromodynamics (QCD) simulations of the fundamental elementary particles that involve a set of gluon and quark fields on a regular four-dimensional lattice. These computations form one of the largest uses of supercomputer time in academic computing.

Loosely synchronous problems are also typically data parallel, but now we allow different data points to be evolved with distinct algorithms. Such problems appear wherever one describes the world macroscopically in terms of the interactions between irregular inhomogeneous objects evolved in a time synchronized fashion. Typical examples, as in Figure 6, are computer or biological circuit simulations where different components or neurons are linked irregularly and modelled differently. Time driven simulations and iterative procedures are not synchronized at each microscopic computer clock cycle, but rather only macroscopically “every now and then” at the end of an iteration or a simulation time step.

Loosely synchronous problems are spatially irregular, but temporally regular. The final asynchronous class is irregular in space and time, as in Figure 3b. A good example is an event driven simulation, illustrated in Fig-
SYNCHRONOUS PROBLEMS

For example:

Microscopic Description of Fundamental Interactions
In Particular, QCD

- Computational structure (almost) identical for all elements in the data domain
- Parallelize by regular partition of data domain
- Run well on SIMD machines
- Message Passing or High Performance Fortran implementation on MIMD machines

Figure 5: The Synchronous Problem Class
Loosely Synchronous Problems

For example: Macroscopic description of physical system in terms of interactions between irregular inhomogeneous objects evolved as a time synchronized simulation. In particular - biological neural network

Parallelize by irregular partition of data domain

**Hardware:**
In general will not run well on SIMD machine.

**Software:**
Initial version of High Performance Fortran cannot describe.

Message passing or extensions of High Performance Fortran on MIMD machines will describe.

Figure 6: The Loosely Synchronous Problem Class
ure 7, that can be used to describe the irregular circuits we discussed above, but now the event paradigm replaces the regular time stepped simulation. Other examples include computer chess [Felten:88] and transaction analysis. Asynchronous problems are hard to parallelize and some may not run well on massively parallel machines. They require sophisticated software and hardware support to properly synchronize the nodes of the parallel machine, as is illustrated by time warp mechanism [Wieland:89a].

Both synchronous and loosely synchronous problems parallelize on systems with many nodes. The algorithm naturally synchronizes the parallel components of the problem without any of the complex software or hardware synchronization mentioned above for event driven simulations. In the original survey, 90% of the surveyed applications fell into the classes that parallelize well. This includes 14% from the embarrassingly parallel classes, and roughly equal (38% each) amounts from synchronous or loosely synchronous class. It is interesting that massively parallel distributed memory MIMD machines that have an asynchronous hardware architecture are perhaps most relevant for loosely synchronous scientific problems.

We have looked at many more applications since the detailed survey in [Fox:88b], and the general picture described above remains valid. Industrial applications have less synchronous and more loosely synchronous problems than academic problems. We have recently recognized that many complicated problems are mixtures of the basic classifications. The first major example with which I was involved was a battle management simulation implemented by my collaborators at JPL [Meier:89a]. This is formally asynchronous with temporally and spatially irregular interconnections between various modules, such as sensors for control platforms and input/output tasks. However, each module uses a loosely synchronous algorithm, such as the multi-target Kalman filter [Gottschalk:90b] or the target-weapon pairing system. Thus, the whole metaproblem consists of a few (≈ 10–50) large grain asynchronous objects, each of which is a data parallel synchronous or loosely synchronous algorithm. This type of asynchronous problem can be implemented in a scaling fashion on massively parallel machines. We call this a metaproblem or asynchronous combination of several synchronous or loosely synchronous problems. A similar example of this asynchronous or embarrassingly synchronous problem class is machine vision and signal processing, where one finds an asynchronous collection of data parallel modules to perform various image processing tasks, such as stereo matching and edge detection. Figure 8 illustrates another example where we outline an approach to designing a new airframe that involves aerodynamics, struc-
ASYNCHRONOUS PROBLEMS

For example:
The world looked at macroscopically in terms of interactions between irregular inhomogeneous objects evolved as an event-driven simulation

**Battle of Hastings**

- Parallelize by “data parallelism” over space of events but no automatic algorithmic synchronization
- Need sophisticated software built on top of message passing between events to ensure synchronization
- Speedup very problem-dependent
- MIMD architectures essential

Figure 7: The Asynchronous Problem Class
tures, radar signature, and the optimization discussed later in Section 5.2. This figure also points out the interesting analogy between heterogeneous metaproblems of class and a heterogeneous computer network.

In the above cases, the asynchronous components of the problems were large grain modules with modest parallelism. This can be contrasted with Otto and Felten’s MIMD computer chess algorithm, where the asynchronous evaluation of the pruned tree is “massively parallel” [Felten88]. Here, one can break the problem up into many loosely coupled but asynchronous parallel components, which give excellent and scalable parallel performance. Each asynchronous task is now a synchronous or loosely synchronous modestly parallel evaluation of a given chess position.

There were a few examples mentioned above of metaproblems in our original survey, but a major part of Table 4, from our New York State activity,
is the Information Integration classification, including manufacturing and the applications 25–33 are essentially all metaproblems. As stated boldly in Table 1, this class is the most important long-term area for HPCC. Further, as in battle management case, many problems that formerly appear asynchronous and were classified in this way in our original survey, are in fact metaproblems. Thus, the parallelism does not come from the difficult (impossible?) asynchronous structure, but the synchronous or loosely synchronous components buried inside the asynchronous shell. Thus, we believe metaproblems and their software support very important.

4 Some Software and Machine Issues

Naturally parallel implementations work “best” if the machine architecture is “similar” to that of the problem. This is summarized in Table 5 where to be precise, success requires that the machine architecture “contains” (is a superset of) the problem architecture. Thus, both SIMD and MIMD machines express synchronous problems, but SIMD machines are typically unsuitable for loosely synchronous problems.

Table 5: What is the “Correct” Machine Architecture for each Problem Class?

<table>
<thead>
<tr>
<th>Problem Class</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>SIMD, MIMD</td>
</tr>
<tr>
<td>Loosely Synchronous</td>
<td>MIMD, maybe SIMD</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>MIMD, but may not perform well without special hardware features</td>
</tr>
<tr>
<td>Compound (Metaproblems)</td>
<td>Heterogeneous network (including World Wide Web)</td>
</tr>
<tr>
<td>Embarrassingly Parallel</td>
<td>Network of workstations MIMD, World Wide Web, sometimes SIMD</td>
</tr>
</tbody>
</table>

Software systems need to be designed so that they can express problems well, and be targeted to relevant machines. Software should not be designed
for a particular machine model—it expresses problem and not machine characteristics.

Table 6: Candidate Software Paradigms for Each Problem Architectures

- **Synchronous**: High Performance Fortran (HPF) [Foster:95a], [HPFCSep:95a], [Koelbel:94a]; Fortran 77D [Bozkus:93a], [Fox:91e], [Hiranandani:92c]; Vienna Fortran [Chapman:92b]; C* [Hatcher:91a;91b]; Crystal [Chen:88b]; APL; Fortran for SIMD parallel computers

- **Loosely Synchronous**: Extensions of the above, especially HPF [Chapman:94b], [Choudhary:92d], [HPF:94a]; and parallel C++ [Bozin:91a], [Chandy:93a], [Grimshaw:93b], [Lemke:92a]; Fortran or C plus message passing [Fox:91m], [McBryan:94a]

- **Asynchronous**: Linda [Factor:90a;90b], [Gelertner:89a]; C/C++ [Chandy:93a]; Time Warp [Wieland:89a]; PCN [Chandy:90a]; WebWork [Fox:95a]

- **Compound Metaproblems**: AVS [Mills:92a;92b], [Cheng:93a]; PCN, Linda (or Trellis built on Linda); Webwork; Fortran-M [Foster:95a]. Generally, extensions of ADA, Fortran, C, or C++ controlling modules written in synchronous or loosely synchronous approach

- **Embarrassingly Parallel**: Several approaches work?
  - PCN, Linda, WebWork, PVM [Sunderam:90a], Network Express [Parasoft:88a], ISIS [Birman:87a;87b;91a]

We have described those issues at length in [Fox:90p;91g;94a], and here we will just present a simple table (Table 6) mapping the five problem architectures into possible software environments. This is presented in a different fashion for HPF and HPC++ in Figure 9 and Table 7, which also points out the distinct runtime support needed for each problem class. One always has a tradeoff between performance and flexibility. Systems listed under
“asynchronous” in Table 6 can typically also be used for synchronous and loosely synchronous problems. As shown in Figure 10, the “asynchronous” software used on loosely synchronous problems will probably provide greater flexibility, but lower performance than software systems explicitly designed for this problem class.

Table 7: Imprecise Mapping of Problem Classes into Runtime and Language Terms

| STATIC Runtime | Synchronous and Embarrassingly Parallel Problems—current HPF |
| ADAPTIVE Runtime | Loosely Synchronous but not Synchronous—future capabilities of High Performance Fortran (HPF+) but can be supported well in message passing |
| ASYNCHRONOUS Runtime | Asynchronous Problems |
| INTEGRATION Runtime and Programming Environments | Metaproblems |
| | AVS works well but also can be integrated into languages such as HPC++, Fortran-M |

Loosely synchronous problems are in some sense the hardest as they have difficult irregularities which must be expressed with high efficiency by the underlying compiler and runtime systems. We, and others, have discussed this at length, both in general [Choudhary:92d;92e], [Fox:90p], [Goil:94a;95a], and in case of High Performance Fortran [Bogucz:94a], [Chapman:94b], [Cheng:94e], [Choudhary:92g;94c], [Fox:94g], [Hawick:95a;95c], [HPF:94a], [HPFapp:95a], [Joubert:95a], [Muller:95a], [Robinson:95a], [Sturler:95a].

Note that Figure 9 refers to “HPF+”—this is some extension, called officially HPF 2 (and later 3 perhaps) of HPF [HPF:93a], [HPFF:95a] to fill gaps in the original language. The current HPF1 handles most synchronous and embarrassingly applications, but requires extension to handle the adaptive irregular data structures typical of loosely synchronous problems.
<table>
<thead>
<tr>
<th>Static</th>
<th>Adaptive</th>
<th>Asynchronous</th>
<th>Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Data</td>
<td>Irregular</td>
<td>Asynchronous</td>
<td>Metaproblems</td>
</tr>
<tr>
<td>Parallel or</td>
<td>Adaptive</td>
<td>e.g., Event</td>
<td>Integrating Static,</td>
</tr>
<tr>
<td>Embarrassingly</td>
<td>Data Parallel</td>
<td>driven</td>
<td>Adaptive, and</td>
</tr>
<tr>
<td>Parallel</td>
<td>e.g., data</td>
<td>simulations,</td>
<td>Asynchronous</td>
</tr>
<tr>
<td></td>
<td>mining,</td>
<td>Expert systems,</td>
<td>modules</td>
</tr>
<tr>
<td></td>
<td>multigrid</td>
<td>Transaction</td>
<td>e.g., Command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>processing</td>
<td>&amp; Control,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ocean -</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Atmosphere</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Integrated climate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>models,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>multidisciplinary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>analysis &amp; design</td>
</tr>
</tbody>
</table>

- Nearly ALL scientific and engineering simulations
- Gives massive parallelism in many metaproblems
- Includes Many NII Applications

Figure 9: General Applicability of HPF, HPF+, HPC++ Classified by Problem Architecture and type of Runtime Support needed
We now quantify these remarks with three case studies, which will link the material of Sections 2, 3, and 4.

5 Machine and Problem Architectures and Appropriate Programming Paradigms in Three Case Studies

We now illustrate the different machine, problem, and software issues with three case studies. These are each broad application areas where there is no one approach. Rather, several very distinct application subclasses are present in each case for which different programming paradigms and machine architectures are appropriate.

5.1 Computational Chemistry and Electromagnetics (Applications 3, 7, and 8)

Many chemistry problems are formulated in terms of states of a chemical system, which can be labelled by an index corresponding to species, choice of wave function, or internal excitation (see Chapter 8 of [Fox:94a]). The
calculation of energy levels, potential or transition probability can often be related to a matrix \(M_{ij}\) whose rows and columns are just the possible system states. \(M\) is often an approximation to the Hamiltonian of the system or it could represent overlap between the states. There are two key stages in such problems:

a) firstly, calculate the matrix elements \(M_{ij}\)

b) secondly, perform one or more of a set of matrix operations

- Matrix Multiplication as in change of basis
- Matrix Eigenvalue determination as in energy level computations
- Matrix Equation solution as in solving multichannel scattering problems

This structure has been elegantly exploited within the "Global Array" programming model built at Pacific Northwest Laboratory [Nicplocha:94a] with a set of tools (libraries) designed for this class of computational chemistry problem.

These two steps have very different characteristics. The matrix element computations a), is of the embarrassingly parallel case as each \(M_{ij}\) can essentially be calculated independently even though subexpressions may be shared between two or more distinct \(M_{ij}\). Each \(M_{ij}\) is a multi-dimensional integral with the computation depending on the details of the states \(i\) and \(j\). Thus, this computation is very time consuming and is not suited for SIMD machines. The natural parallel algorithm associates sets of \((i,j)\) with each node of a parallel computer. There are some relatively straightforward load balancing issues and essentially no internode communication. Thus, a MIMD cluster of workstations with modest networking is sufficient for this step a). The final matrix manipulations have quite a different character. These synchronous problem components are suitable for SIMD machine and often required substantial communication so that a workstation cluster will not be effective. Matrix multiplication could be exception as it is insensitive to latency and communication bandwidth for large matrices and so suitable for workstation clusters.

One of the standard approaches to computational electromagnetics (CEM) is the method of moments [Harrington:61a;67a;68a], [Jordon:69a]. This is a spectral method, which rather than solving the underlying partial differential equation (Maxwell’s), expands the desired solution in a set of “moments”.

29
This leads to a similar situation to that described above for computational
chemistry where \(i\) and \(j\) label moments for CEM and not the chemical state
[Cheng:94a;94c]. Note that in both cases, the matrix \(M\) is treated as full
[Cheng:94c], and is quite different from the familiar sparse matrices gotten
from discretizing a partial differential equation. We note in passing that
such spatial discretization is a quite viable approach to CEM and leads
to a totally different computational problem architecture from the spectral
moment formulation.

HPF can handle both stages of the matrix based CEM or chemistry
problems [Robinson:95a]. The matrix solution stage exploits fully the For-
tran 90 array manipulation and clearly requires good compiler support for
matrix and vector manipulation primitives. NPAC’s experience with a pro-
duction CEM code PARAMOM from the Syracuse Research Corporation is
illuminating [Cheng:94c]. Both stages could be implemented on IBM SP-2
with specialized Fortran code for the matrix element generation joined to
SCALAPACK based matrix solution [Choi:92c]. However, the CM-5 im-
plementation was not so simple. The CMSSL library provided exceptional
matrix solution with good use being made of the CM-5’s vector nodes. How-
ever, the matrix element computation was not so straightforward. Perfor-
mance on the CM-5 nodes was poor and required conversion of the original
Fortran 77 to Fortran 90 to both exploit the vector nodes and link to
CMSSL. However, whereas the Fortran 90 notation was very suitable for
matrix manipulation, it is quite irrelevant for the matrix element generation
stage—as already explained, this exploits the INDEPENDENT DO and not the
array notation for explicit parallelism. Thus, we split the PARAMOM code
into a metaproblem with two sub-problems corresponding to the two stages
discussed above. Now we implemented each stage on the most appropriate
architecture. The “embarrassingly parallel” Fortran 77 matrix element gen-
eration stage was run on a network of workstations, the equation solution
stage used the optimized libraries on the CM-5 or SP-2. The linkage of
these stages used AVS, but one could alternatively use many other coor-
dination software approaches. We expect to test our use of World Wide Web
technology WebWork [Fox:95a] on this example.

This simple example illustrates three problem classes: embarrassingly
parallel, synchronous and metaproblems, and associated machine and soft-
ware architecture. There is an interesting software engineering issue. Typ-
ically, one would develop a single Fortran program for such a computa-
tional chemistry or electromagnetics problem. However, better is separate
modules—in this case, one for each of two stages—for each part of prob-
lem needing different parallel computer treatment. In this way, we see the breakup of metaproblems into components, and use of systems such as AVS as helpful software engineering strategies [Cheng:92a;94d]. We have successfully used such an approach to produce an effective parallel version of the public domain molecular orbital chemistry code MOPAC [MOPAC:95a].

Not all chemistry computations have this structure. For instance, there is a set of applications such as AMBER and CHARMM that are based on molecular dynamics simulations, as described in Chapter 16 of [Fox:94a], [Ranka:92a]. These are typically loosely synchronous problems with each particle linked to a dynamic set of "nearest neighbors" combined with long-range nonbonded force computations. The latter can either use the synchronous $O(N_{\text{particle}}^2)$ algorithm or the faster, but complex loosely synchronous fast multiple $O(N_{\text{particle}})$ or $O(N_{\text{particle}} \log N_{\text{particle}})$ approaches [Barnes:86a], [Edelsohn:91b], [Goil:94a], [Goil:95a], [Greengard:87b], [Salmon:90a], [Singh:93a], [Sunderam:93a], [Warren:92b], [Warren:93a].

5.2 Computational Fluid Dynamics and Manufacturing (Applications 1, 2, 3, 4, and 32)

CFD (Computational Fluid Dynamics) has been a major motivator for much algorithm and software work in HPCC, and indeed extensions of HPF have largely been based on CFD (or similar partial differential equation based applications) and molecular dynamics [Bogusz:94a], [Choudhary:92d;94c], [Dincer:95b], [Goil:94a;95a], [Hawick:95a;95b], [HPF:94a]. Partial differential equations can be quite straightforward on parallel machines if one uses regular grids, such as those coming from the simplest finite difference equations. However, modern numerical methods use either finite elements or a refinement strategy for finite elements, which gives rise to irregular meshes. Approaches, such as domain decomposition and multigrid, also give use to complex data structures. From a Fortran programmer's point of view, simple finite differences can be well described by Fortran array data structures. Corresponding parallelization of such applications is well suited to the current HPF language, which is centered in decomposing arrays. All the more advanced partial differential equation schemes naturally need somewhat more sophisticated (than simple arrays) data structures, including arrays of pointers, linked lists, nested arrays, and complex trees. The latter are also seen in fast multipole particle dynamics problems, as well as fully adaptive PDE's [Edelsohn:91b]. Some excellent methods, such as the Berger-Oliger adaptive mesh refinement [Berger:84a] require modest
HPF extensions as we have shown in our Grand Challenge work on colliding black holes [Haupt:95]. However, as Saltz's group has shown in a set of pioneering projects [HPF:94a], many important PDE methods require nontrivial HPF language extensions, as well as sophisticated runtime support, such as the PARiT [Saltz:91b] and CHAOS systems [Edjali:95a], [Hwang:94a], [Ponnusamy:93c;94b]. The needed language support can be thought of as expressing the problem architecture (computational graph as in Figure 3(a), which is only implicitly defined by the standard (Fortran) code. Correctly written, this vanilla Fortran implies all needed information for efficient parallelism. However, this information is realized in terms of the values of pointers and cannot be recognized at compile time for either static or compiler generated dynamic runtime parallelism. This fundamental problem is of course why Fortran is a more successful parallel language than C as latter naturally uses pointer constructs that obscure the problem architecture even more. The runtime support for PDE's must cope with irregular and hierarchical meshes and provide the dynamic alignment decomposition and communications optimization that HPF provides for arrays.

Now, let us consider manufacturing as a major industrial application of PDE's. Here, HPCC offers an important opportunity to build futuristic manufacturing systems allowing customizable products with integrated design (conceptual and detailed), manufacturing process, sales and support. This scenario—sometimes called agile manufacturing—implies other major thrusts including concurrent engineering and multidisciplinary analysis and design. Here, we can use an example where NPAC is working under NASA sponsorship with the MADIC (Multidisciplinary Analysis and Design Industry Consortium) collaboration involving Rockwell, Northrop Grumman Vought, McDonnell Douglas, General Electric, General Motors and Georgia Tech. We are in particular involved in establishing for this NASA project, the NII requirements for a future concurrent engineering concept called ASOP (Affordable Systems Optimization Process). Aircraft are now built by multi-company collaborations with international scope which virtual corporation needs collaborative and networked software engineering and workflow support. Configuration management is a critical need. ASOP links a range of disciplines (from manufacturing process simulation, electromagnetic signature, aeronautics and propulsion computation linked to CAD databases and virtual reality visualization) using MDO—multidisciplinary optimization—techniques. The integration of conceptual (initial) and detailed design with the manufacturing and life cycle support phases naturally requires the integration of information and computing in the support sys-
tem. WebWork [Fox:95a] has been designed for this. Further, we see this problem is a heterogeneous metaproblem with perhaps up to 10,000 (Fortran) programs linked together in the full optimization process. This is basically an embarrassingly parallel meta-architecture with only a few of the programs linked together at each stage. The program complexity varies from a full PDE simulation to an expert system to optimize location of an inspection port to minimize support costs. So efficient parallel solution of PDEs is part, but not all of the support needed for manufacturing. HPCC will only have major impact on manufacturing when it can support such heterogeneous metaproblems, including large scale database integration.

5.3 Monte Carlo Methods (Applications 4, 6, 7, 9, 11)

We have already mentioned in Section 3, Quantum Chromodynamics Simulations as a classic example of large scale Monte Carlo simulations suitable for parallel machines. As described in Chapter 4 of [Fox:94a], this application is straightforward to parallelize and very suitable for HPF as the basic data structure is an array. The array represents a regular structure in space time as seen in the simplest finite different problems. The Monte Carlo occurs at each grid point and is typically local (nearest neighbor) so that the overall problem architecture is just like that of a PDE. This specific computation is from an academic field, but is typical in structure of some practical material science problems. Further, just as many PDEs have irregular data structures, the same is true of many Monte Carlos. QCD is typical of simulations of crystalline substances with a regular array of atoms. However, many substances—in particular gases and liquids—have irregular particle distributions and many of issues discussed briefly in Section 5.2 for finite element methods. As described in Chapter 14 of [Fox:94a], there is a subtle point that distinguishes Monte Carlo and PDE algorithms as one cannot simultaneously update in Monte Carlo, sites with overlapping neighbors. This complicates the loosely synchronous structure and can make problem architecture look like that of a synchronous event driven simulations—here events are individual Monte Carlo updates. “Detailed balance” requires that such events be sequentially (if arbitrarily) ordered. In the example of [Johnson:86c] described in [Fox:94a], a clever implementation gave good parallel performance.

Monte Carlo methods can be implemented quite differently—above we decomposed the underlying physical data. One can also use “data parallelism” on the random number set used in the simulation. This is not possi-
ble for QCD for two reasons. Firstly, the physical dataset is so large it would not fit in the memory of a single node—we need to decompose the physical dataset just to get enough total memory. More importantly, one can run QCD with several different starting points. However, all Monte Carlos—using importance sampling of the Metropolis type employed by QCD—have a “thermalization stage” where one must get “into equilibrium” before the sampling is useful. Thermalization is very time consuming for QCD and makes multiple starting points of limited value. However, there are many cases where this is not true, and as show in Chapter 7 of [Fox:94a], one can get an embarrassing parallel architecture for Monte Carlo problems. Each instance of the problem has the full physical dataset, but can be run independently with different random number streams. Like many such embarrassingly parallel cases, the different instances do need to accumulate their data—in this case, Monte Carlo averages. One important examples of this class of application is Quantum Monte Carlo used in many ab initio chemistry problems [Kalos:85a].

Yet, a different set of issues comes with a class of Monte Carlo problems which are termed “clustered.” In most physical system Monte Carlos, one updates a single “entity” (grid point or particle) at a time. This is very ineffective when there is substantial correlation between neighboring points. A simple example comes from ferromagnetic materials where domains form where spins are locked in the same direction over large regions. Clustering algorithms are quite hard to find for sequential systems, and their parallelization is challenging and very different from the earlier examples. As discussed in Section 12.6 of [Fox:94a], the algorithm is similar to that used in region finding in image processing [Copty:93a;94a;95a]. Parallelism requires consideration (as in domain decomposition for PDEs) of inter and intra region issues.

5.4 Summary

Each of three case studies illustrates how different applications and different numerical approaches to a given problem, lead to very different problem architectures and correspondingly the needed software support. Although our discussion is not complete, we do think that it is quite typical, and that a similar situation is seen in the other applications of Table 4, and summarized in the last two columns.
References


36


[Choudhary:92g] Choudhary, A., Fox, G., Haupt, T., and Ranka, S. “Which applications can use high performance Fortran and FortranD—industry standard data parallel languages?,” in Proceedings of Fifth


[HPFapp:95a] [http://www.npac.syr.edu/hpfa/algorithms.html](http://www.npac.syr.edu/hpfa/algorithms.html). A collection of applications designed to test HPF, which is online at NPAC.


