UCSD CS Technical Report #CS95-451

Zoom: A Hierarchical Representation for Heterogeneous Applications

Cosimo Anglano*
Dipartimento di Informatica
Università di Torino

Jennifer Schopf, Rich Wolski, and Francine Berman†
Dept. of Computer Science and Engineering
University of California, San Diego ‡

January 5,1995

Abstract

Heterogeneous network computing is defined as the implementation of a large, complex application on a network of possibly diverse computers. With the increase in network communication speeds and the availability of fast workstations and multiprocessors, heterogeneous network computing is emerging as a viable option for the development of performance-efficient applications. In this paper, we describe Zoom, a hierarchical representation in which heterogeneous applications can be described. The goal of Zoom is to provide an abstraction that computer and computational scientists can use to describe heterogeneous applications, and to provide a foundation from which program development tools for heterogeneous network computing can be built. Three levels (structure, implementation and data) of the Zoom hierarchy are described and are used to illustrate two heterogeneous applications. Extensions to Zoom to include additional resource parameters required by program development tools are also discussed.

1 Introduction

Heterogeneous network computing is defined as the implementation of large, complex applications on distributed networks of computers. It is an important and emerging field at the junction of computer and computational science. Heterogeneous network computing has evolved as a natural result of four key trends in computation.

First, in the last two decades, no single universal parallel architectural model has emerged. Successful commercial multiprocessors have efficiently supported a limited number of programming models. Pipelined parallelism, data-parallel, task-parallel, SPMD, systolic and dataflow computations have been introduced. However, with each new architectural type, only a limited set of programs is able to achieve maximal efficiency.

Second, the increasing complexity of high-performance computing systems has sharpened the focus on architecture-specific libraries and software reuse. For example, chemistry and physics codes have utilized very efficient generic equation solvers to yield cost-effective performance. Many scientists employ different combinations of previously-written algorithms, each of which addresses

*Partially supported by the ESPRIT—BRA project No. 7269 "QMIPS" and by the Italian CNR project "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo", grant 91. 00879. PF69

†Supported in part by NSF grants ASC-9301788, ASC-9308900 and funding from the U. C. San Diego Academic Senate.

‡Email addresses of the authors are mino@di.unito.it and {jenny, rich, berman}@cs.ucsd.edu.

a different aspect of the problem. The same "scientific" code may be coupled with different "visualization" systems, for example, each of which presents a different view of the resultant data.

A third and critical trend is that network communication speeds are increasing with respect to the processing power provided by a single machine. A cluster of multiple computers connected by a fast network is becoming an increasingly effective computational platform. Not only can more computational power be brought to bear on a single problem, but existing resources may be combined in different, more dynamic ways to meet the needs of a given application. Moreover, network computing provides a way to revitalize older equipment. In many cases, new performance may be gained using older equipment with which programmers have experience, familiarity and better software support.

The fourth trend is that data reuse is becoming essential. Data collected for different applications may be combined profitably to address new problems. Since the data "belongs" to different applications, it may be unalterably partitioned into sections, each of which may only be accessed by a unique set of machines. For example, the Calcrust application, which provides a 3-dimensional image of the earth's surface and its crust, combines USGS data, sounding data taken by various oil companies, and NASA satellite data [5]. In each case, the data was collected for some other purpose and each "owner" has a specialized system for accessing it.

At the junction of these trends is the emergence of heterogeneous network computing. Since faster networks reduce the overhead resulting from the use of multiple machines, heterogeneous networks can provide more processing power and, in many cases, more performance to programmers. Successful heterogeneous programming, however, calls for a sophisticated strategy.

To develop performance-efficient heterogeneous applications, many constraints must be balanced. Ideally, portions of the application that exhibit distinct computational paradigms should be implemented on architectures which support them in the most performance-efficient manner. In addition, pipelining between program components may be required to amortize network communication time. Scientists may decrease time spent in software development and promote performance by reusing code that has been optimized to a particular machine. Furthermore, data previously stored at different sites may force parts of the application to execute on different machines, since moving data may not prove profitable or even possible. Balancing these constraints demonstrates the difficulty of programming heterogeneous networks, but the potential benefits for heterogeneous applications are clear.

In this paper we present Zoom, a representational framework that captures the structure of heterogeneous applications and the interactions between their components. Zoom has two goals. First, it serves as an abstraction by which computer and computational scientists can communicate about the salient features of heterogeneous applications. Second, it provides a base representation on top of which program development tools for heterogeneous applications can be built. We chose the name Zoom to emphasize the hierarchical nature of the framework, allowing application components to be captured at a high level, or to be "zoomed" in to a more detailed level as the needs of the implementation dictate.

The next subsections describe related work and a prototype heterogeneous application to provide motivation for the Zoom representation. Section 2 describes the semantics and structure of the Zoom representation. Section 3 uses Zoom to represent two heterogeneous applications. Sections 4 and 5 discuss extensions to the current representational framework and present conclusions.

1.1 Related Work

As heterogeneous network computing is a new and evolving field, there are few antecedents to Zoom. The closest comparison can be drawn with the HeNCE representation [4] used to specify heterogeneous applications as part of a graphical tool for PVM [20]. Both HeNCE and Zoom allow the programmer to describe the components of an application and the type of communication between those components. There are some differences, however.

First, HeNCE is a "flat" representation, i.e. all application components are specified with the same level of detail. Zoom is hierarchical and provides an increasing amount of detail at each level. Some of these details (e.g. data conversion costs) are not represented within the HeNCE model. Second, each HeNCE node represents a subroutine written in either C or Fortran. At Level 1, Zoom nodes represent coupling units – logical components of the application for which communication remains within the confines of a single machine. Each component may be written in any language which can be implemented on the target machine. Zoom explicitly captures the relationships between between subroutines that must always be mapped together, and those that may communicate across machine boundaries, whereas HeNCE components are machine-independent and are intended to be scheduled based on a user-defined cost matrix.

The HeNCE representation is also designed to work specifically with PVM. Zoom has taken a different approach. The representation is network and interface independent and was developed to serve as a language of discourse between scientists and programmers as well as the basis for software tools. HeNCE provides an execution model whereas Zoom does not, i.e. Zoom provides a way to describe the structure of applications but there is no notion of state and state transitions of the represented application. We have explored a Zoom-to-HeNCE translation which would enable programmers to use the Zoom hierarchical representation and the HeNCE execution model [2]. Because of distinct features in each representation, not all programs can be translated this way, but the process gives insight into how both Zoom and HeNCE might be expanded to be compatible.

Paralex [1] is a system similar to HeNCE, but with further restrictions. Its ability to represent pipelining is constrained to a single program graph, and each procedure is allowed only a single output. Further, it is targeted to, and relies upon, the Isis [7] parallel programming toolkit, whereas Zoom is independent of a particular programming environment.

1.2 A Prototype Heterogeneous Application

To motivate key aspects of heterogeneous applications, we describe an application based loosely on the Calcrust application. Calcrust [5], developed at JPL, provides a 3-dimensional rendered image of the earth's surface and its crust. The input data comes from several different sources (oil company archives, USGS surveys, daily satellite sweeps, etc.) The data is first filtered and refined via an FFT based set of routines. Data from the filtering stage then flows to a renderer which provides a complete image.

For the purposes of illustration, we expand the Calcrust application to include an explicit smoothing stage. In addition, we hypothesize the existence of multiple implementations for the Filter, Smoother and Render stages[1]. In practice, multiple algorithms are often available on a given machine and researchers choose among them for a particular implementation.

The filtering stage of the application feeds inputs to both the smoothing stage and the rendering stage. The smoothing stage feeds only the rendering stage, and the output of each complete iteration is a rendered image. The FFT stage is highly vectorizable and has been implemented for

[1] We have expanded Calcrust to demonstrate the major characteristics of most heterogeneous applications. We will use this example in the next section to illustrate of the graphical features of each of the levels of Zoom.

two different vector supercomputers. Also, two different smoothing algorithms are included, one for a workstation, and one for a massively parallel SIMD machine. Finally, three different rendering implementations are provided, each implementing a different algorithm, but all targeted to a MIMD machine. Note that the renderer (regardless of the algorithm selected) may be pipelined with the other application stages.

This example illustrates several important general characteristics of heterogeneous applications.

- *Parallelism may be exploitable within individual application components.*

The smoothing stage, for example, may be executed by a SIMD algorithm on a SIMD machine or by a sequential algorithm on a workstation, or possibly both. Any choice, however, yields a correct execution of the overall application.

- *Communication may be overlapped with computation in the form of pipeline parallelism.*

If two application stages are independent, they may execute in parallel on two different machines. If one stage depends on another (due to a data dependency) but can compute using partial inputs, then the two stages can be overlapped, forming a pipeline. It is this non-strict form of communication which enables heterogeneous applications to tolerate long network latencies.

- *Different algorithms may be used to implement the same stage of an application.*

The different characteristics of various architectures may make it necessary to use different algorithms for the same application stage, depending on the desired mapping. In the example, the smoothing stage may either be executed on a SIMD machine or on a sequential workstation. This does not imply, however, that an identical algorithm will be used on both. Rather, the application consists of alternative algorithms for the smoothing stage, each of which is tuned for a specific architecture. It may also be that alternative algorithms, each emphasizing a different aspect of the overall problem, are available. The three rendering algorithms, for example, may achieve different levels of accuracy with respect to color, shading, texture mapping, etc. They are, however, interchangeable depending on what the user wishes to emphasize.

- *Data may need to be converted between stages.*

Data may need to be converted from one format to another as it is communicated between stages within the application. For example, in the Calcrust application, the vector machines that execute the filtering stage may use a different floating-point format than the MIMD machine executing the rendering stage. The cost of converting the data from one format to another must be accounted for in the execution time of an application.

Aside from exposing general heterogeneous characteristics, the example also illustrates a key point. Even for the simplest heterogeneous application, many important features are obscured or implied. In the description of the example, data dependencies are not easily discerned, nor it is clear where data independence exists between application components. Further, the notion of data conversion is implied by the different architectural possibilities, but it is not explicit. From the perspective of implementation, additional detail is important so a "hierarchical" way of expressing the application – at first with a simple structure, and later with more details relevant to the implementation – is indicated.

In the next section, we introduce the Zoom representation. In subsequent sections, we use Zoom to represent current heterogeneous applications.

2 The Zoom Representation

The Zoom representation provides a language of discourse that seeks to consistently and concisely capture the salient features of heterogeneous applications. Zoom is hierarchical; each successive level presents a greater level of detail.

Level 1, the *structure level*, depicts an application as a sequence of phases, each of which describes a set of loose relationships between logical components of the program. Primary consideration is given to the relationships between components which cross machine boundaries. In particular, the granularity of the decomposition described at this level is very coarse. Preliminary or informal discussions of heterogeneous applications generally abstract the problem to this level which is also useful when developing decompositions or considering future enhancements.

Level 2, the *implementation level*, focuses on implementation information relevant to the computational scientist. Different implementations of algorithms and a summary of the type of communication between components is made explicit and depicted by different graphical features. Some data conversion requirements are represented.

Level 3, the *data level*, focuses on a more detailed description of the algorithms and their data requirements. This level exposes some attributes of the implementation costs relevant for the design of program tools. Format and structure data conversions are made explicit, as are feasible configurations of machines. The Level 3 Zoom representation provides information that depicts the relationship between communication and computation in enough detail to expose potential sources of parallelism and possible performance tradeoffs.

2.1 Level 1

The structure level (Level 1) of the Zoom representation depicts an *application* as a linear sequence of *phases*[2]. Each phase circumscribes a graph whose nodes are *coupling units* and whose edges are dashed lines representing communication between coupling units. Coupling units represent logical components (collections of program tasks) of the application that can potentially communicate across machine boundaries. Communication between program tasks that does not span machine boundaries is not made explicit at Level 1.

A Level 1 representation consists of *phase boundaries* that enclose a portion of the application, *boxes* representing coupling units labeled with their logical component, and *dashed arcs* representing communication between coupling units and to and from phase boundaries. Note that dashed arcs indicate that although communication takes place, the form of the communication is not specified, i.e. they do **not** dictate a precedence relationship. Dashed arcs simply show that at some point in time communication takes place. For example, computation and communication may be overlapped.

Phases may be executed once or repeatedly. Repeated phases of the application are enclosed by sets of double lines, one at the beginning of the repeated phase and one at the end. Phases that are executed once are enclosed by a set of single lines, again, one at the beginning of the phase and one at the end. Both single and double line pairs are called *phase boundaries* and the portion of the application between phase boundaries is called the *phase body*. Coupling units that are destination points of arcs which emanate from the opening phase boundary are called *input coupling units* for that phase. Coupling units that are source points of arcs terminating at the closing phase boundary are called *output coupling units* for that phase.

The first phase of an application may have a single arc leading to the opening phase boundary

[2]The decomposition of heterogeneous applications into phases represents the structure of typical heterogeneous programs and follows [19].

emanating from a key word *INITIAL*. The last phase of an application may have a single arc emanating from the closing phase boundary to a key word *RESULT*.

Any machine implementation, data, or resource management issues for the coupling units are unspecified at Level 1. A Level 1 representation for the Calcrust application appears in Figure 1. In this application, there is one phase which is not repeated. The *Filter* coupling unit is an input coupling unit and the *Render* coupling unit is an output coupling unit.

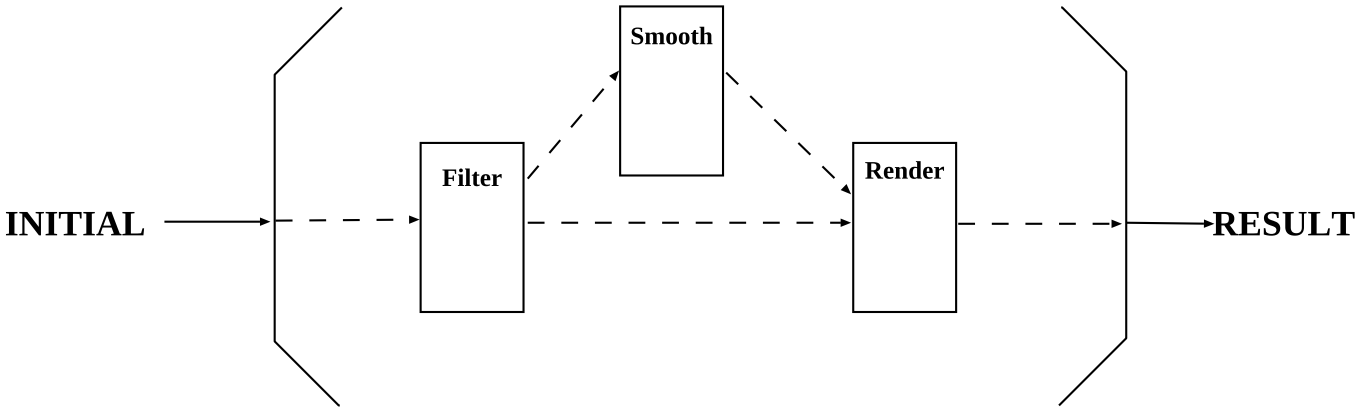

Figure 1: Level 1 Representation of Calcrust Application.

2.2 Level 2

The implementation level (Level 2) provides more detail about the algorithms, possible machine implementations, and communication requirements of the application. Before we describe the components of Level 2 more formally, we briefly outline their function. At Level 2, coupling units contain *ovals* which represent distinct algorithm-machine pairings. Each oval represents a different combination of algorithm, language, programming paradigm, machine, etc. Ovals may in turn contain graphs whose nodes are *circles* representing program tasks which will be executed on a given machine for a given implementation. Level 1 dashed arcs are replaced by *wires* indicating strict communication between all possible implementations of the source and destination coupling units, or *tubes* indicating that some pair of implementations will communicate using an overlapping or pipelined communication.

In addition, Level 2 includes data conversion information. A *square* intersecting a tube or wire indicates that the type or structure of the data communicated along that edge must be converted for some pair of implementations (ovals) in the source and destination coupling units. Finally, the location of termination criteria for repeated phases is provided through the use of *triangles*.

We describe each of these graph features more completely in the following subsections.

2.2.1 Ovals

At Level 2, one or more *ovals* are embedded in the coupling units from Level 1. Each oval represents a distinct implementation of an algorithm on a machine. Ovals are labeled with the machine to which they correspond since the algorithm is essentially named by the coupling unit itself. All ovals in the same box are logically equivalent – they all implement the logical component represented by the coupling unit. If there are different implementations of the component (for different architectures or the same architecture), each will be represented by its own oval. Note that an oval is assigned to a single machine, and that the different ovals within the same coupling unit do not communicate with one another in the same iteration. They are data independent.

2.2.2 Circles

Circles represent tasks within the logical component implemented by an oval. Both circles (tasks) and communication between them in the form of solid arcs may be drawn inside an oval. This graph of circles is used to describe in greater detail those portions of the application which make up that logical component and do not cross machine boundaries. For convenience, if an oval has only one circle, it is omitted.

2.2.3 Arcs

The Level 1 dashed arcs change in Level 2 according to the type of communication. A computation is *strict* with respect to communication if it makes its outputs available only at its conclusion, or if it cannot begin until all of its inputs are available, or both. It is *non-strict* otherwise. For example, assume that a coupling unit produces a grid as its output. If it produces the entire grid at the end of its execution, the communication of this output is strict. If it produces the grid a row at a time, sending it in pieces to the receiver coupling unit, it is non-strict.

Depending on the communication relationships between two coupling units, a Level 1 dashed arc becomes either a *wire*, represented as a continuous line, or a *tube*, represented as two parallel lines. Tubes summarize the configuration in which some pair of communicating computations may overlap their execution and communication, i.e. overlapped communication *can* happen. Alternatively, wires denote communications which are strict. In this case, *all* pairings of ovals between adjacent coupling units communicate strictly. In practice, a wire between two coupling units denotes a precedence relation between them within the context of a single iteration.

Recall that with the exception of phase boundaries, all arcs have source and destination coupling units. *Input* tubes and wires emanate from the opening phase boundary and terminate at input coupling units within the phase body. *Output* tubes and wires emanate from output coupling units and terminate at the closing phase boundary of the phase that contains the coupling units.

During phase repetitions, input couping units are assumed to be receiving data from some set of output coupling units within the same phase, although possibly from different previous iterations. That is, data entering the closing boundary is assumed to be available at the opening boundary during some successive iteration(s). More detail about communication is given in Level 3.

2.2.4 Octagonal Boxes

In the general case, a given coupling unit (represented as a box at Level 1) may have several implementations (represented as Level 2 oval nodes). When the application is executed, one or more of the ovals within a given box must be selected. If no execution or the simultaneous execution of multiple ovals is allowed, the coupling unit is represented as an *octagonal box* in Level 2. Alternatively, coupling units in which only a single oval may be selected for execution are represented by rectangular boxes at Level 2.

2.2.5 Triangles

The location of the termination criteria for a repeated phase is indicated by the position of a *triangle* associated with the phase. If the triangle is placed over one of the circles or ovals within the phase body, the circle or oval is assumed to contain some computation which determines the termination of the phase. If the triangle is placed to the lower right of the closing phase boundary, the calculation is determined by some predefined criteria not computed within the phase body (e.g. a loop index variable).

It is also important to consider conversion costs once implementation information is known. Consider the Level 1 representation shown in Figure 1, if a Cray is executing the filtering stage and a Paragon is executing the smoothing stage, then every communication between these stages requires some conversion of the data [3]. We have identified two features of the data that may need to be converted : *format,* and *structure. Format* refers to the different ways individual architectures represent the simplest data types. For example, most high-performance computing systems support IEEE "standard" floating point format, and a faster, higher precision, but non-portable native format. Floating point data communicated between two machines using different formats must be converted.

A *structure* change refers to a change in the data structures used by ovals within different coupling units. For example, data manipulated as a FIFO queue by an algorithm in one coupling unit may need to be converted to a binary tree for use by an algorithm in another coupling unit. Alternatively, two communicating algorithms may apply the same semantics to a data set, but each may implement those semantics differently. A matrix, for example, may be implemented as an array by an algorithm in one coupling unit, and as a linked list in another.

Format conversion depends only on the primitive data types used by the coupling units and the architectures to which each is assigned. Since such conversions are fundamentally dictated by the various architectures employed by an application, we represent them as part of Level 3. We include structure conversion in Level 2 (as structure *squares*) since structure conversion influences the execution cost of the application with respect to a given implementation.

2.2.6 Squares

Squares, which intersect tubes or wires, represent structure conversion operations required between some pair of ovals in adjacent coupling units. The time needed to perform the conversion is a function of the machine(s) where the conversion takes place.

Conversion routines are computations, and as other computations, must be assigned to a machine in order to be executed. If the structure conversion computation will always be performed at a particular coupling unit, the location of the structure conversion square can be used to provide additional information. In the Level 2 Zoom representation, the structure conversion square may be either placed at the source or destination coupling units to indicate that the conversion will always be performed by the implementations within that coupling unit.

The square may also be divided into halves at both the source and destination coupling units if an intermediate representation or canonical form will be used, or placed in the middle of the arc connecting them if this information is unknown. Also, if the structure conversion square can belong to one of several pairings of ovals in adjacent coupling units, then it is placed in the middle of the tube or wire to act as summary information.

Figure 2 provides a Level 2 description of the Calcrust application. Note that the smoothing coupling unit can be run on either the front-end of a Thinking Machines CM-2 (marked WS), or the back-end (marked CM-2), or both, and that more than one instance can be executed at a given time, represented by an octagonal box. The three different possible Paragon implementations for

[3] A general form of data conversion was discussed for heterogeneous applications in [14] as well as [8].

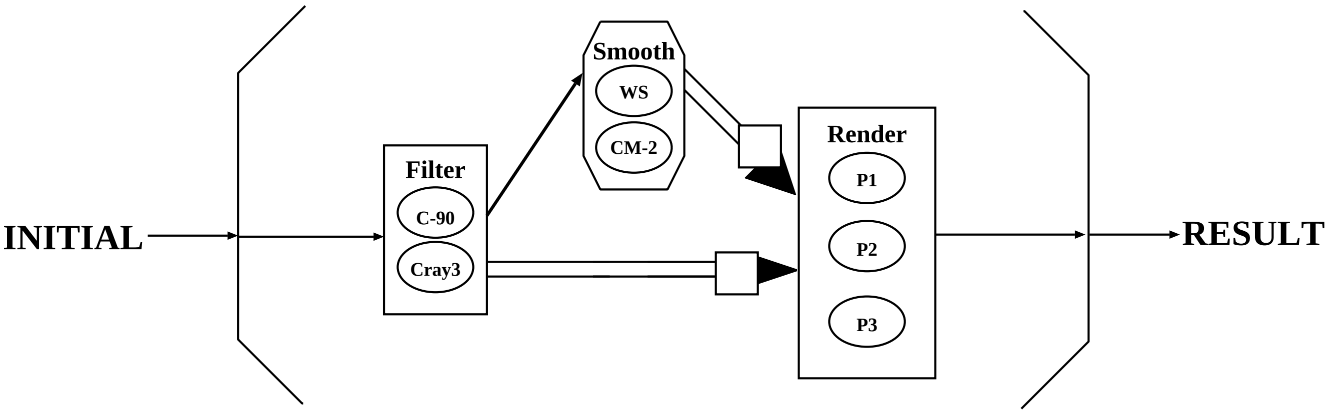

Figure 2: Level 2 Representation of the Calcrust Application.

the renderer are marked P1, P2 and P3. For the purposes of illustration, we hypothesize structure conversion between the Filter and Renderer and between the Smoother and Renderer, both of which are executed on the Paragon.

2.3 Level 3

At the data level (Level 3), more specific information about communication and an application's resource requirements is depicted. In particular, communication granularity, structure conversion requirements, and format conversion requirements are shown for each legal pairing of implementations (Level 2 ovals). Such information is necessary for program development tools and the more accurate cost models required for optimization and scheduling.

At Level 3, each Level 2 tube or wire is augmented with a set of three *association matrices* showing *connectivity, format conversion* and *structure conversion.* The structure conversion matrix replaces the squares from Level 2. Connectivity matrices particularize tubes and wires between coupling units or between a coupling unit and a phase boundary from Level 2. For tubes or wires between coupling units, every oval in the source box corresponds to a row in each association matrix and every oval in the destination box corresponds to a column. Individually, the association matrices are defined as:

- **Connectivity Matrix** An element (i, j) of the connectivity matrix (denoted by a "c" located next to its lower right hand corner) is a pair of integers (x, y) where x is the size of the data item sent across the communication link from oval i in the source coupling unit to oval j in the destination coupling unit, and y is the total amount of data. The fraction x/y represents the granularity of the communication and is less than 1 if the communication is pipelined or overlapped and equal to 1 if the communication is strict. If there is no communication between two ovals, we place $(0, 0)$ as its connectivity matrix entry. Note that using this representation, wires can be thought of as a special case of tubes in which $x = y$.

  The fraction x/y is intended to capture the proportion of the overall data structure communicated at one time between coupling units. One quarter of an image, for example, might be sent from the Filter to the Smoothing phase in the Calcrust application. In this case, the relevant entry in the connectivity matrix would have value 1/4. If Zoom was utilized as the graphical representation for a program development tool, however, it might be reasonable to substitute other measures such as the number of bytes or words transmitted for these ratios.

- **Format Conversion Matrix** The format conversion matrix is denoted by an "f" near its lower right hand corner. If a pair of ovals require a format conversion, then their corresponding element (i, j) is marked with a S if the conversion is executed at the source oval, D if the conversion is executed at the destination oval, B if the conversion is done on both ends (i.e. as when an intermediate representation or canonical form is used), and N if no format conversion occurs or the ovals do not communicate.
- **Structure Conversion Matrix** Structure conversions between ovals at (i, j) are marked S if the conversion computation is executed at the source oval, D if it is executed at the destination oval, B if the conversion computation is split between both ends, and N if no structure conversion occurs or the ovals do not communicate. The structure conversion matrix is marked with an "s" next to its lower right hand corner.

The association matrices for output tubes or wires (those linking output coupling units to phase boundaries) specify something slightly different than those located between coupling units. For a repeated phase, the matrices describe the connectivity, structure and format between the ovals within the source output coupling unit and the ovals in *all* input coupling units in the next iteration of that phase. If connections do not exist, an N appears in the table. For a non-repeated phase, *no* matrices are specified for output tubes and wires. In both repeated and non-repeated phases, *no* association matrices are specified for input tubes and wires.

To define connectivity, format and structure conversion between phases, association matrices are inserted between consecutive phase boundaries. There is a row in every association matrix for each oval of each output coupling unit in the *ith* phase and a column for each oval of each input coupling unit in the $(i+1)st$ phase for all i. As before, an N appears in the table if no connection exists.

Although the connectivity matrix enforces one communication type (tube or wire) between an output coupling unit in a phase and an input coupling unit in the next phase, the Level 3 graphical description can be used to provide additional information. If, for example, the output edge of phase i is a tube and the input edge of phase $i+1$ is a wire (this is summarized as a tube in the connectivity matrix), the figure is interpreted to indicate that archival storage or buffering is required between phases. The same mechanism may be used between iterations of a repeated phase to indicate storage patterns. Note that all such communications are summarized in the connectivity matrix using tubes. While the formalism must account for all cases, in practice the number of phases and ovals is typically quite small, and the representation does not become excessively unwieldy.

The Level 3 description of the Calcrust application appears in Figure 3. No association matrices appear on the tube to the closing phase boundary since the phase is not repeated. For illustration, we hypothesize that the Cray 3 and CM-2 do not implement consecutive components. We also suppose that 1/4, 1/2 and all of the input is sent at one time for the three different Render implementations on the Paragon. Note also that since the data representation used by the Paragon for floating point numbers is different from that used by the Cray, a format conversion is necessary. In the example, we depict these conversions as taking place on the Paragon. Similarly, the necessary data conversion between the Cray and the CM-2 is assigned to the Cray in the figure.

2.4 Using Zoom

During interactions with various applications scientists, it became apparent that different development groups were finding it difficult to exchange ideas and leverage existing work. Zoom provides scientists with a domain of discourse – a way to communicate concisely about heterogeneous applications. An example of this utility can be seen with the General Circulation Model application

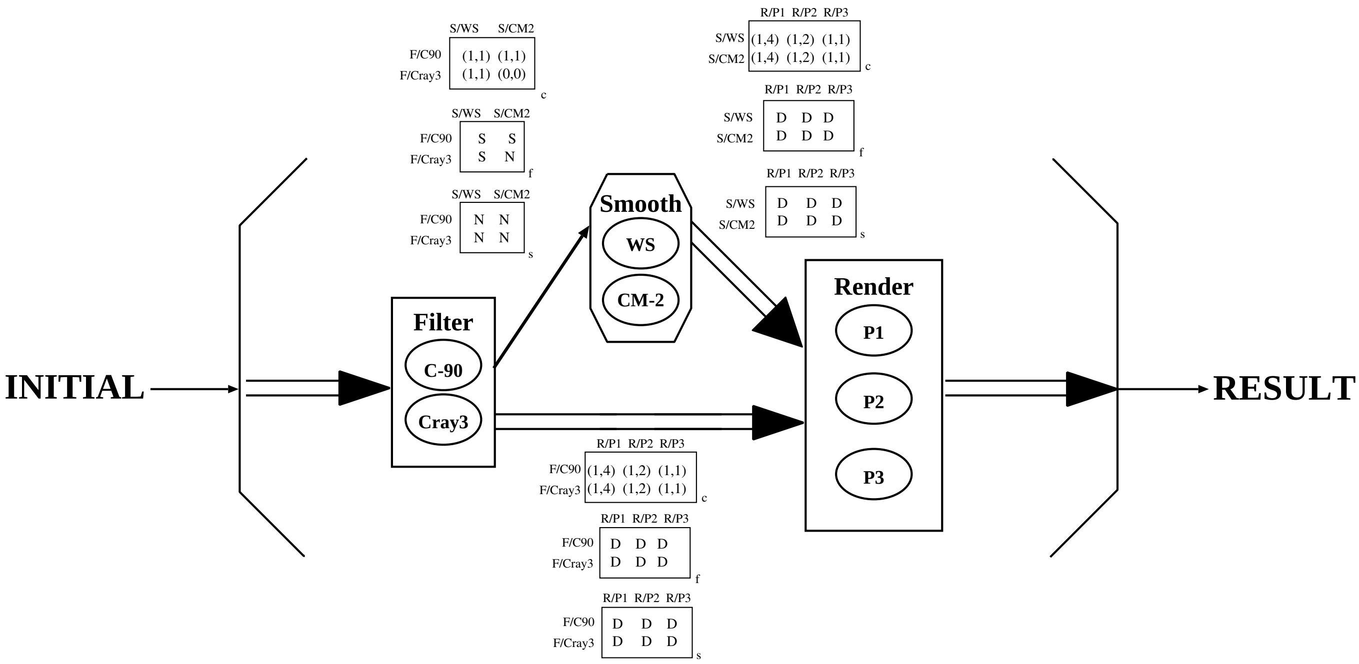

Figure 3: Level 3 Representation of the Calcrust Application. Labels X/Y on the association matrices rows and columns are algorithm-machine matchings. For example, F/C90 stands for the Filter algorithm used on the C90.

discussed in Section 3.2. Two groups have taken different approaches to the same application. In contrasting their approaches, Zoom provides a level of detail from which the underlying differences become clear.

We intend to develop Zoom as an interface for a set of tools designed to aid in the development and management of applications on a heterogeneous network of computers. Information about the application, supplied by the programmer, will be combined with static and dynamic system level information to make realistic performance estimates. We intend to use Zoom as an interface to a control system with live feeds from the machines on a network. A user would be able to make educated decisions about the running state of the application, and could change the mapping or scheduling choices to better suit the current network conditions. We are developing this strategy as the basis for application-based dynamic scheduling methods.

Zoom can also be used as a design and development aid as it reveals important details about possible heterogeneous implementations. As a paper study, the differences in design alternatives can be made explicit. In addition, if the Zoom representation was tied to an on-line performance prediction system, more performance-efficient design choices could be made during the development of the application.

3 Examples

In this section, we use the Zoom representation to describe two heterogeneous applications.

3.1 CASA 3D-REACT Application

Quantum mechanical reactive scattering (3D-REACT) is a chemical application that is used to predict the energy levels of various chemical reactions from first principles. At the California Institute of Technology, physical chemists M. Wu and A. Kuppermann, as a part of the Casa Project [5] have been working on a 3D-REACT application that simulates a hydrogen-deuterium reaction of $H + D_2 \Longrightarrow HD + D$.

The hydrogen-deuterium reaction is studied because it is one of the simplest reactions that can be calculated from first principles. Wu and Kuppermann have refined this calculation by including a property known as the geometric phase [24]. Michael Berry (University of Bristol) first called attention to this phase, which now bears his name, in a variety of physical systems [6]. The inclusion of this phase alters the calculation so that an attribute of symmetry is used to to achieve a more precise result with respect to the basic laws of quantum dynamics. For more details on the reaction itself, see [15].

3.1.1 General Structure of the Application

Wu and Kuppermann believe that the computing power offered by a heterogeneous configuration of machines can be used effectively to reduce the wall clock time needed by this computationally intensive application [22, 23]. As part of the CASA project, they have investigated using a Cray C-90 and an Intel Delta in tandem to implement this application. The 3D-REACT application lends itself to such a heterogeneous implementation because it has several well-defined computational parts, whose characteristics allow the overlapping of communication with computation.

The application, which essentially calculates the solution to a six-dimensional Schrödinger equation, has three logical components. The first logical component, the local hyperspherical surface function (*LHSF*) calculation, generates a set of five-dimensional LHSFs for the expansion of the time-independent Schrödinger equation. The most time-consuming and memory-intensive step of this section is the diagonalization of fully symmetric matrices. Sets of eigenpairs are used to obtain a set of dense symmetric matrices, called propagation matrices.

The second logical component performs a logarithmic derivative propagation (*Log-D*), given a set of LHSFs and propagation matrices. This operation involves a large number of matrix inversions, and the resulting matrices are stored on disk.

The third logical component reads the matrices (stored on disk) resulting from the second logical component and performs an asymptotic analysis (*ASY*) for each one.

Since the ASY logical component is not computationally intensive, it is grouped with the Log-D logical component to form a single coupling unit, labeled *LogD/ASY*. Note that the ASY logical component may direct the entire computation (LHSF and then LogD/ASY) to be repeated if termination conditions are not met.

When 3D-REACT is executed on a single machine, the application is set up so that all of the LHSFs are calculated at one time. The number of surface functions (LHSFs) is an input parameter. Then the entire Log-D phase is executed. The asymptotic analysis (ASY logical component) decides whether enough calculations have been performed, and the application either ends or another complete set of surface functions is considered.

On a heterogeneous system, the problem is subdived into smaller subdomains of 5 to 20 surface functions per subdomain to mask the latency of communication. First one machine calculates the LHSFs for a given subdomain, then the data is passed to a second machine which calculates the Log-D portion of the problem for that subdomain. While this second machine is calculating the first subdomain, the first machine can start calculating the second subdomain, and in this way communication and computation are overlapped. After all subdomains are considered, the ASY section determines whether another full set of surface functions needs to be calculated.

3.1.2 The Zoom Representation

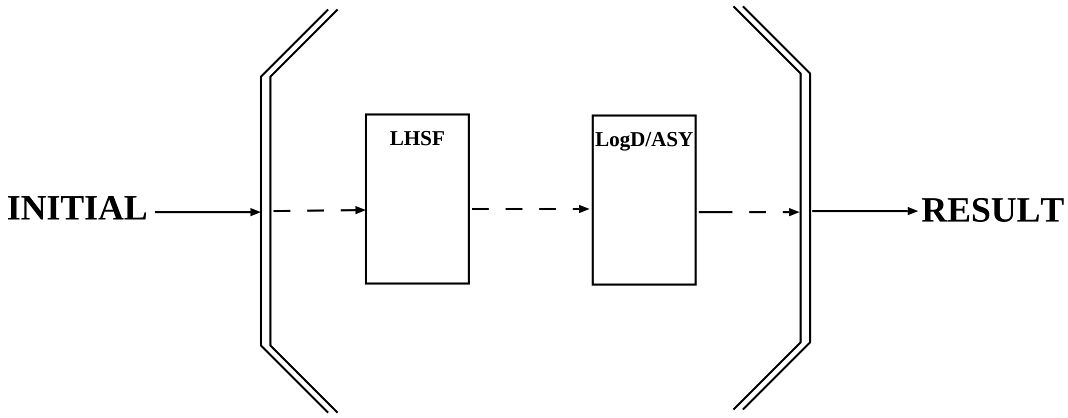

Figure 4: CASA Chemical Reaction, Level 1 Representation

In Figure 4, we show the Level 1 representation for the 3D-REACT application. The two coupling units for this calculation are shown as two boxes, indicating the design choice of incorporating the Log-D and the ASY logical components into a single coupling unit. The dashed line in between the coupling units indicates that information is communicated from the LHSF coupling unit to the Log-D/ASY one. By definition, the communication model is not specified; the dashed arcs merely indicate that communication takes place.

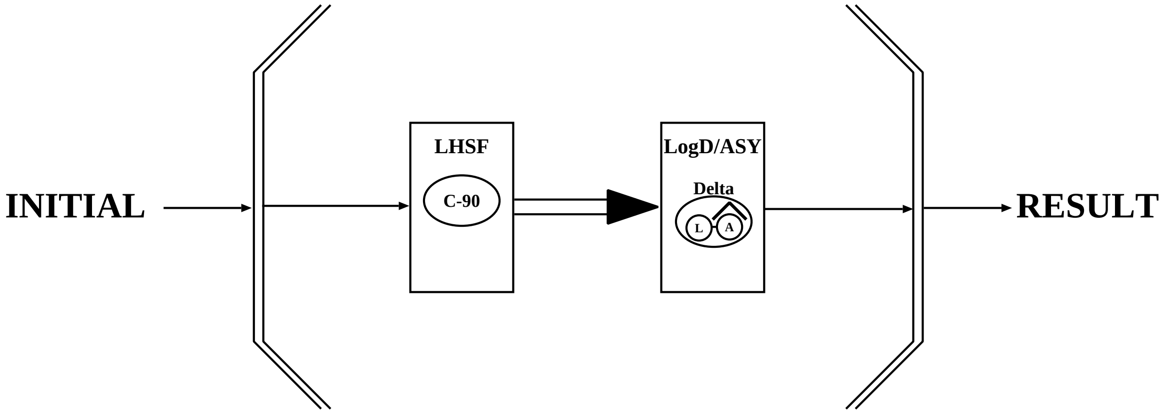

Figure 5: CASA Chemical Reaction, Level 2 Representation.

The Level 2 Zoom representation is shown in Figure 5. An oval in the LHSF box indicates that code has been developed for this section of the application on the C-90. The Log-D/ASY box contains a single oval, representing the code developed for the Delta. Circles are used to distinguish the Log-D (L) from the ASY (A) parts with termination criteria indicated (by a triangle) as performed by the ASY calculation. The pipelined nature of the communication is represented by means of a tube. There is no structure conversion required between LHSF and LogD/ASY for this application.

Figure 6 shows the Level 3 Zoom representation. The association matrices indicate the granularity of pipelining on the tube. Format conversions are performed within a phase iteration and between phase iterations on the Delta.

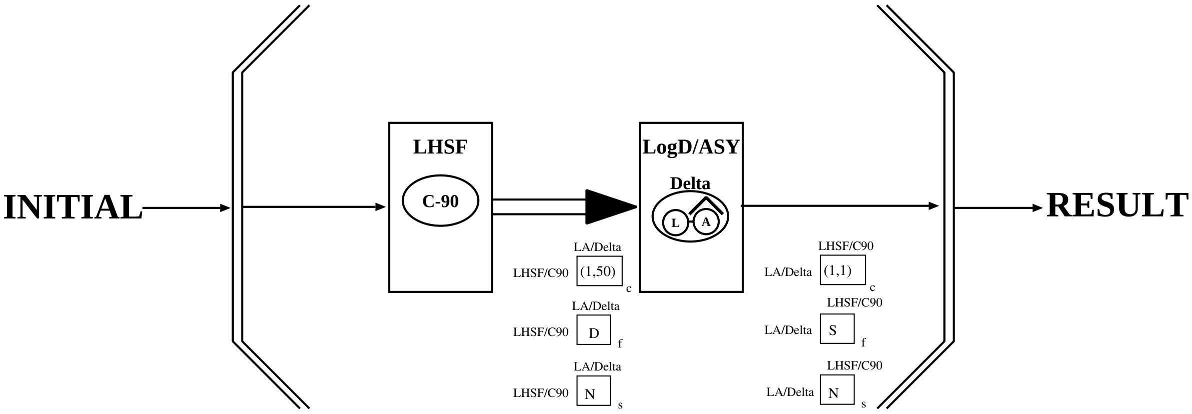

Figure 6: CASA Chemical Reaction, Level 3 Representation.

3.1.3 Extensions

The 3D-REACT application is currently partitioned so that the Log-D and the LHSF coupling units execute for approximately the same amount of time to achieve an effective load balancing. However, this partitioning is not essential to the program's correctness or efficiency. One extension currently under development is to add a second phase to the application so that more Log-D calculations can be performed. This phase would allow more energy levels to be calculated. Note that these calculations are completely independent and can be executed in parallel without any communication.

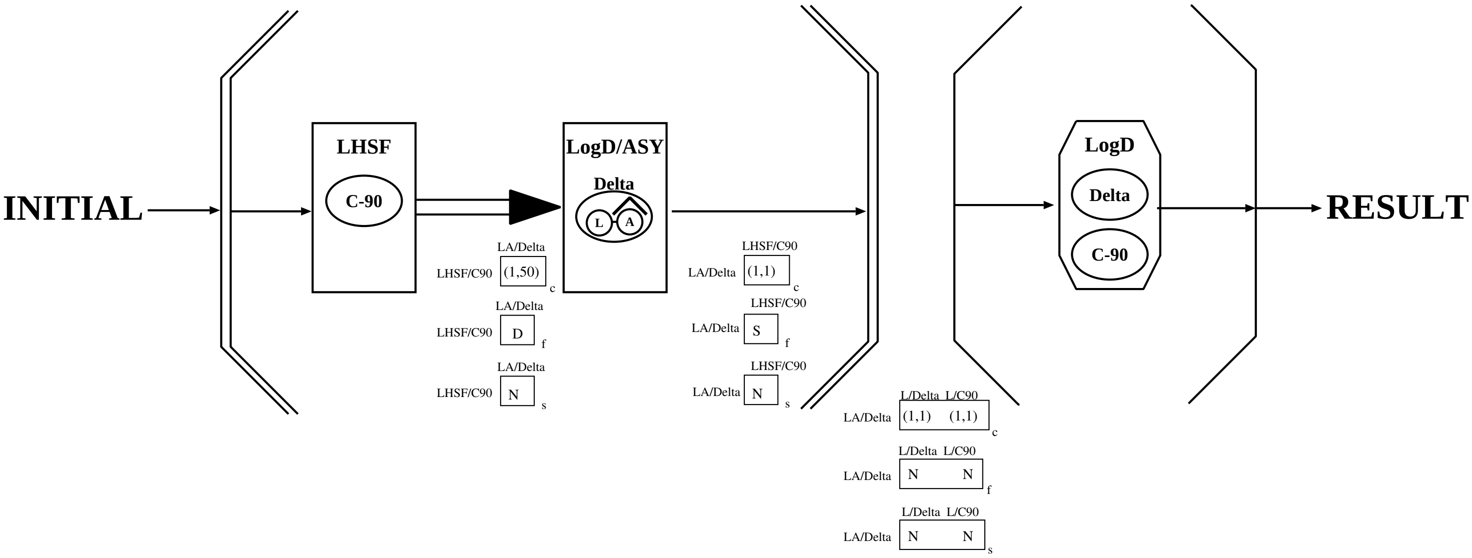

Figure 7: CASA Chemical Reaction, Level 3 Representation of Two Phase Extension.

Currently, once a full set of surface functions (LHSF calculations) and the corresponding number of Log-D calculations have been performed, the ASY computation determines whether the calculation should stop or not. Another version of the application would include further iterations of LogD, so that instead of stopping after the final ASY, both the C90 and the Delta can begin to perform LogD calculations for different sets of energy levels, completely independently of one another. This second phase would have no interprocessor communication since after the last surface function is calculated, both machines have a full set of LHSFs stored in their respective memories.

As seen in Figure 7, the first phase of the new version is identical to the original application shown in Figure 6. A second phase has been added that consists of only the Log-D logical component, with implementations for both the C-90 and the Delta. Note that since both the C-90 and the Delta implementations can be executed at the same time, the coupling unit is octagonal. Also note that there is no communication between the Delta and the C90 during this phase, other than an initial signal and a set of parameters communicating to each machine which energy levels it has to calculate.

3.2 The General Circulation Model

In the study of the earth's climate, General Circulation Models (GCMs) prove to be amongst the most useful tools. Generally, they solve the equations governing fluid motion on a rotating sphere, but most allow for the introduction of other physical effects (cloud convection, turbulent mixing, etc.) as parameters. Through this parameterization, an individual GCM modeling the atmosphere can be coupled with one modeling the ocean to study how the interaction between the atmosphere and the ocean affects global climate [18].

Until recently, most GCM codes have been heavily optimized for vector computing environments. They tend to be computationally intensive grid-oriented calculations that are well-suited to data parallel algorithms. The current generation of microprocessor-based distributed-memory multiprocessors offer larger memory systems and greater theoretical performance limits than vector or shared-memory vector systems. Much of the current GCM development activity, then, centers on parallelization for multiprocessor computers [11, 13]. Gigabit networks linking various high-performance computing platforms [5] renders it possible to use several different machines in the execution of a single GCM, or a coupled collection of GCMs. By dividing the codes into communicating components, each of which is well-suited to a different computational paradigm, heterogeneous computing on a fast network promises improved performance. The atmospheric GCM developed at UCLA, for example, is composed of highly vectorizable dynamics routines, and easily parallelizable physics routines [16]. By mapping the vector routines to a vector computer, and the parallel routines to a parallel computer, connected by a high-speed network, good overall execution performance should be possible.

3.2.1 Comparative Study

Potentially, several GCM simulations can be coupled to yield a comprehensive climate model. As a first step toward realizing such a system, independent groups based at UCLA and the Lawrence Livermore National Laboratory (LLNL) are currently formulating coupled atmospheric and ocean simulations. The UCLA group has been focusing on minimum execution time performance (and in particular, achieving superlinear speedup) using a heterogeneous suite of machines [16, 17]. By decomposing the problem into a repeated phase whose execution can be pipelined, the communication latencies can be effectively masked. The goal of the UCLA effort is to achieve the shortest possible execution time given a finite set of diverse computing resources.

Alternatively, the Livermore group has been focusing on portability across a wide variety of platforms (at the possible expense of performance). To ensure the longevity of a code such as a coupled GCM, rapid portability to new architectures is necessary. Further, cluster computing using networked sets of workstations promises cost-effective compute cycles. The LLNL approach has been to use FORTRAN and PVM [20] as a portable language environment. Since PVM is supported for both workstation networks and multiprocessors, the Livermore GCM implementation will run on both a cluster or workstations and more tightly coupled computing platforms. They have not yet considered how to best leverage the resources in a network of distinct machine types.

Clearly, the goals of these two groups are complementary even though they have each chosen to focus on different measures of performance (execution time versus porting time). The Zoom representation for both approaches serves as an analysis tool exposing their similarities and differences.

3.2.2 Coupled GCM Description

The atmospheric GCM (AGCM) has been developed by Professor A. Arakawa and his collaborators over several years at UCLA [3]. It predicts horizontal velocity components, potential temperature, water vapor and ozone mixing ratios, surface pressure, and ground temperature. The UCLA AGCM also predicts the depth of the planetary boundary layer (PBL) which it treats as well-mixed. The code can be parameterized with cumulus convection and its interactions with the PBL, solar and radiative heating, and orographic gravity wave drag [18].

The AGCM is organized into two relatively well-defined components:

- AGCM/Physics, which diagnostically computes the effects of motion processes resolved by the model. The results produced by AGCM/Physics are supplied to the second component (AGCM/Dynamics) as forcing terms in hydrodynamic equations.

- AGCM/Dynamics prognostically computes the evolution of fluid flow according to the primitive flow equations.

The computational requirements for AGCM/Physics (Ap) are typically an order of magnitude greater than that for AGCM/Dynamics (Ad). However, to obtain a better load balance, multiple Ad executions can be made using the results of a single Ap execution. In the current implementation AGCM, each Ap execution simulates an hour of real time. The results for the simulated hour are then used as inputs for 8 successive executions of Ad, each of which simulates a successive 7.5 minute interval. The entire computation is then repeated to simulate a time period of hours, days, or longer.

The OGCM (O) component of the model is based on work done by K. Bryan and M. Cox at the NOAA Geophysical Fluid Dynamics Laboratory, Princeton University [9, 12]. It predicts horizontal velocity components, temperature, salinity, and optionally other tracers. Density is determined from temperature and salinity using either Knudsen's equation [10] or the UNESCO formula [21]. The model's top is assumed to be a rigid lid. Velocities are split into components corresponding to vertically averaged flow and the deviation from the vertical average. Calculation of the vertical average requires only adjacent grid-point values in both the horizontal and vertical directions. The deviation calculations, however, require the solution to an elliptic boundary-value problem that uses nonlocal data. Under the current model, the local calculation precedes the global solver.

In the coupled model, the AGCM executes for a predefined simulated time period. At the conclusion of this interval, it passes time averaged wind stress, heat, and water flux data for the interval to the OGCM. It then uses this information to simulate the same time interval for the corresponding region of ocean returning the sea surface temperature to the AGCM. The grid densities used by the AGCM and the OGCM are different, however, so the data must be interpolated and aggregated as it passes between the two coupled models. Note that this structure conversion is independent of the platforms used to execute any portion of the total application.

In terms of the components of each model, data dependencies exist between Ap and Ad, between Ap and O, but not between Ad and O. That is, the dynamics routines within the AGCM require the forcing field information computed by the physics routines. Similarly, all of the parameters required by the OGCM are also completely computed by Ap. Therefore, Ad and O are independent and may be executed concurrently.

3.2.3 Level 1 Description

The Level 1 Zoom representation shown in Figure 8 captures the top-level relationship between the coupled components of the model.

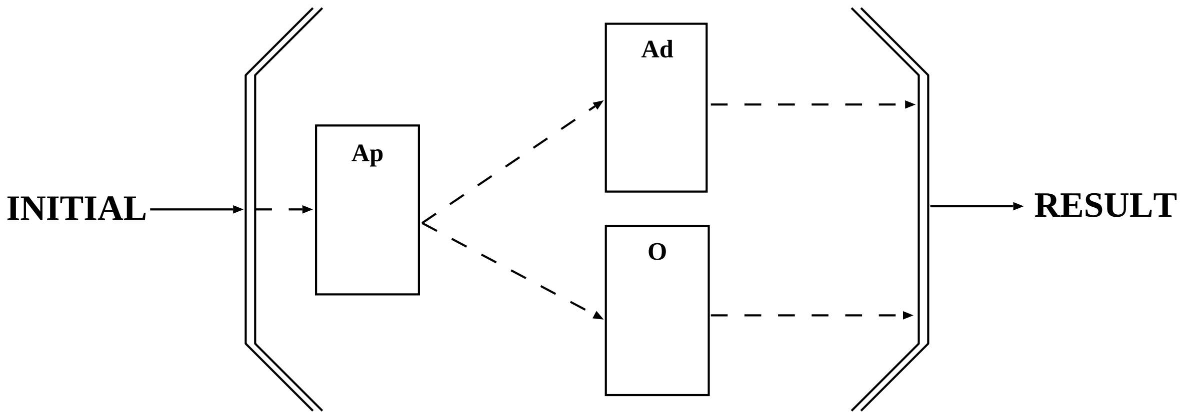

Figure 8: Coupled Global Climate Model, Level 1 Representation.

The iteration boundaries delineate the simulation interval. By executing the Ad and O components in parallel, a reduction in wall-clock execution time is possible, however, the components may be executed linearly in any order that does not violate the data dependencies. At Level 1, Zoom simply depicts the coupling of the components and not their potential mappings. Further, it is important to note that at this level, both the LLNL and UCLA models are the same.

3.2.4 Level 2 Description

At Level 2, the differences between the UCLA approach and that taken by Livermore start to become apparent. The UCLA model employs overlap within a timestep between Ap and Ad, and also between Ap and O. The cycle within the Ad coupling unit indicates that multiple (8) Ad calculations are executed during each simulated interval. In [18], the authors discuss the concept of I/O decomposition in which Ad and O can begin computing their values for a given timestep before Ap has completely computed all of its values for the same timestep. In this way, Ap, Ad, and O can all execute concurrently. Ap and Ad form a pipeline, Ap and O form a separate pipeline, and Ad and O are independent. Further, the UCLA implementation consists of codes optimized for different machine types available via the CASA high-performance testbed. Ap and O have been coded for a message-passing MIMD architecture type while Ad has been optimized for vector computing. The Level 2 Zoom representation of the UCLA code is depicted in Figure 9. Note that

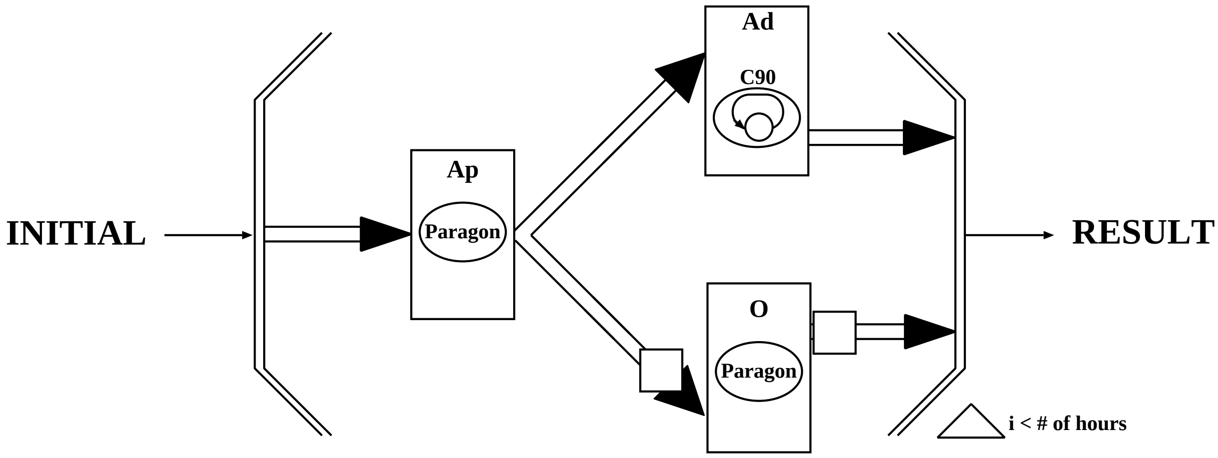

Figure 9: UCLA Implementation of the Coupled GCM.

all structure conversions (required because of different grid densities) are performed by the ocean coupling unit. Also note that the number of iterations is determined by the parameter specifying the number of hours the simulation will be run.

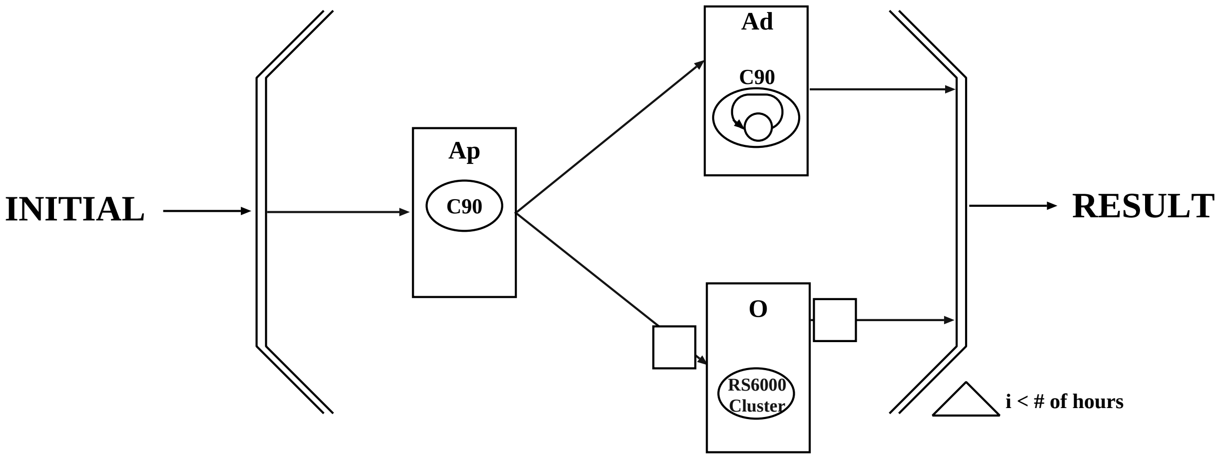

Figure 10: Level 2 Representation of the LLNL Implementation of the Coupled GCM.

Figure 10 shows the Level 2 representation of the Livermore implementation. Livermore's initial target is a cluster of RS6000 workstations and a C90, hence the machine configuration is different. At the time of this writing, only the OGCM has been converted to run in parallel on the cluster. The AGCM components are coupled but execute solely on a C90. All communication in the model is strict as shown by the wires between components. That is, there is no pipeline parallelism in the Livermore implementation – only data-independent parallelism between Ad and O. Additionally, all necessary structure conversions are performed with O on the RS6000 cluster.

3.2.5 Level 3 Description

Level 3 for UCLA and LLNL applications are shown in Figures 11 and 12. Note that for the UCLA implementation, format conversions are handled at both source and destination of the (Ap,Ad) and

(Ad,Ap) tubes because Express is used. Similarly for the Livermore implementation, PVM handles format conversions between machines.

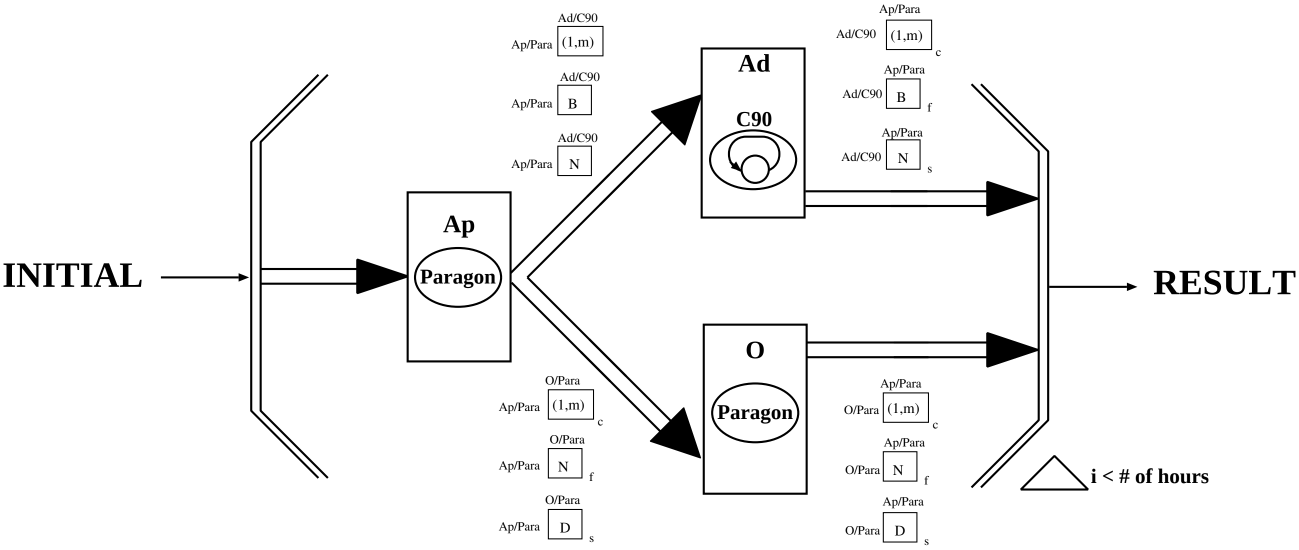

Figure 11: Level 3 Representation of the UCLA Implementation of the Coupled GCM. m = 1, 3, 4, 5 or 8 depending on the implementation.

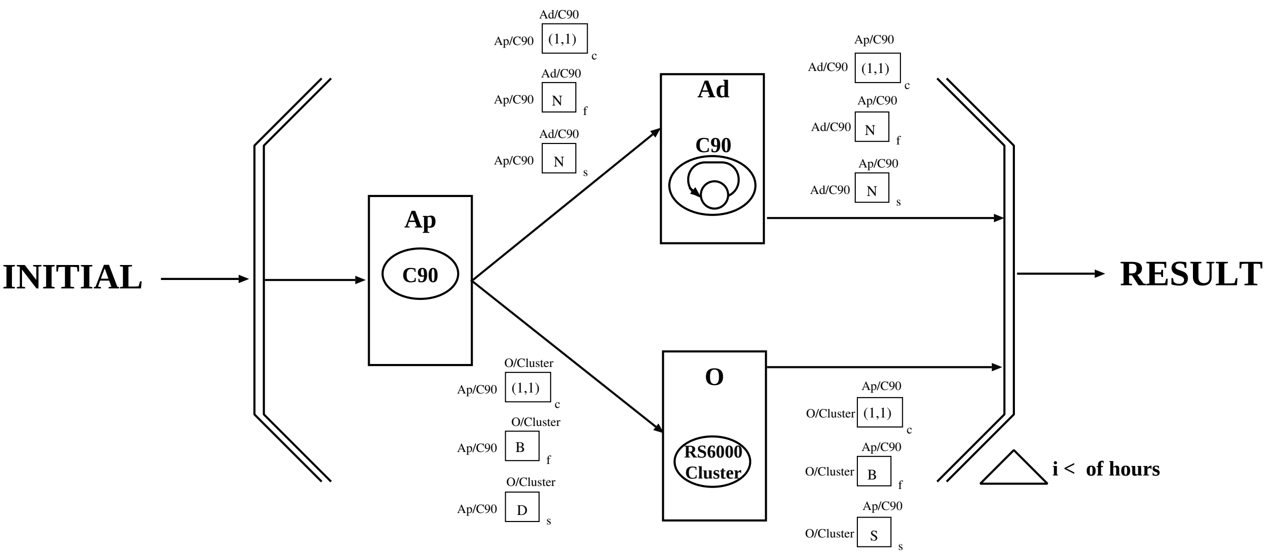

Figure 12: Level 3 Representation of the LLNL Implementation of the Coupled GCM.

4 Extensions to the Zoom Representation

Although levels 1, 2 and 3 of the Zoom representation provide information about implementation choices and communication, they require more detail if they are to be used as an interface for programming tools. Processor and memory capacities, for example, that would be of interest to an automatic scheduling system are not shown at Levels 1, 2 or 3. Such detailed information, essential to integrating the Zoom representation with any performance or mapping tool, must be included at Level 3 or higher. We have included only part of this information at the current Level 3. We believe that further study will be needed to determine whether additional information should be represented textually or graphically, and whether the best representation is as part of Level 3 or at a higher level. The process of tool development should help further define the Zoom representation.

In the next subsections we outline some ideas on extending the current formulation of Zoom.

4.1 Resource Requirements

The ovals at Levels 2 and 3 specify machine-algorithm pairs and provide a coarse idea of communication granularity, however only basic resource requirements are specified. In particular, an application may require certain features or capabilities from the machine to which it is mapped. The number of processors, the amount of memory required on each processor, the estimated execution time, whether the processors and network are dedicated or non-dedicated, and the I/O requirements are all parameters relevant to scheduling and the accurate analysis and prediction of performance. We expect that a tool will allow expansion of the oval graphical feature to depict this information either graphically or textually. As we consider further heterogeneous applications and progress in the design of program tools, Level 3 will be expanded and additional levels may be developed to represent relevant and useful resource and application information.

4.2 Conditional Executions and Timing Information

One peculiarity of heterogeneous applications is that different implementations of the same coupling unit may be provided either for the same or for different machines. During successive executions it may be necessary or convenient to run different implementations of a given coupling unit, or possibly more than one implementation of a given coupling unit at a time. Most frequently, only one implementation is active at a time, however. We represent these two cases in Levels 2 and 3 by using two different features, namely rectangular and octagonal boxes.

Consider now the issue of determining which implementation will be executed. A particular implementation might be chosen, for example, on the basis of the size of the input data set, or as a consequence of the choice of another instance (in the same or a different coupling unit). In some cases, combinations of implementations may be incompatible, or the execution of a particular implementation may only be triggered under a specified condition. In this situation we say that the heterogeneous application contains *conditional executions*. Details about these conditions must also be spelled out clearly as a part of the Zoom representation so that necessary information can be provided for any on-line tools.

An additional question concerns the introduction of simulation time steps into the representation. In the GCM example, I/O domain 1 at time step t coming from Ap is used by Ad to produce the corresponding data for I/O domain 1 at t as well as parts of I/O domain 1 at times t+1 through t+k inclusively. It would be natural to label each I/O domain with its number and time step in this application.

5 Conclusion

In this paper, we have described the Zoom representation for heterogeneous applications, and demonstrated its use. Zoom has two goals: to be used as a means for communicating with computational and computer scientists about heterogeneous applications, and as a basis for a set of program development and performance tools for heterogeneous network computing. Such tools will prove critical to the development of the full potential and power of heterogeneous network computing.

It is our belief that the most useful work in heterogeneous network computing will be based on the real needs of heterogeneous programmers. Zoom has been developed in this spirit, and the structures and components provided in the representation are a result of many discussions with designers and programmers of heterogeneous applications. We are currently carrying this work further to develop software tools based on the Zoom representation which will assist in the complex task of performance-efficient heterogeneous programming.

6 Acknowledgements

We wish to thank all the application scientists who took time to talk to us and who gave us direction: Scott Baden, Larry Bergman, Pete Eltgroth, John Farrara, Scott Kohn, Aron Kuppermann, Celeste Mataratzzo, C. Roberto Mechoso, Reagan Moore, Carl Scarbnic, Joseph Spahr, Dan Stanfill, Paul Stolorz, and Mark Wu. We are also grateful to the Heterogeneous Reading Group at UCSD for their useful comments on an earlier version of this paper.

References

[1] Alvisi, L., Amoroso, A., Baronio, A., Babaoglu, O., et al. Parallel scientific computing in distributed systems: the paralex approach. In *Computer and Information Sciences VI. Proceedings of the 1991 International Symposium* (October 1991), M. Baray and B. Ozguc, Eds., vol. 2, pp. 1093–103.

[2] Anglano, C., Wolski, R., Schopf, J., and Berman, F. Developing heterogeneous applications using zoom and hence. *to appear in the Proceedings of the Heterogeneous Computing Workshop* (1995).

[3] Arakawa, A., and Lamb, V. R. Computational design of the basic dynamical processes of the ucla general circulation model. *Methods in Computational Physics 17* (1977), 173–265.

[4] Beguelin, A., Dongarra, J., Geist, G., Manchek, R., Plank, J., and Sunderam, V. Hence: A user's guide version 1.2. Tech. Rep. CS-92-157, University of Tennessee, February 1992.

[5] Bergman, L., Braun, H.-W., Chinoy, B., Kolawa, A., Kuppermann, A., Lyster, P., Mechoso, C. R., Messina, P., Morrison, J., Stanfill, D., St.John, W., and Tenbrick, S. Casa gigabit testbed : 1993 annual report; a testbed for distributed computing. Tech. Rep. CCSF-33, Caltech Concurrent Supercomputing Facilities, May 1993.

[6] Berry, M. Anticipations of the geometric phase. *Physics Today 43*, 12 (December 1990), 34–40.

[7] Birman, K., and Marzullo, K. Isis and the meta project. *Sun Technology 2*, 3 (Summer 1989), 90–104.

[8] Bisiani, R., and Forin, A. Multilanguage parallel programming of heterogeneous machines. *IEEE Transactions on Computers 37*, 8 (August 1988), 930–45.

[9] Bryan, K. A numerical method for the study of the circulation of the world ocean. *Journal of Computational Physics 4*, 3 (October 1969), 347–76.

[10] Bryan, K., and Cox, M. D. An approximate equation of state for numerical models of ocean circulation. *Journal of Physical Oceanography 2*, 4 (October 1972), 510–14.

[11] Chervin, R. M., and A. J. Semeter, J. An ocean modeling system for supercomputer architectures of the 1990's. In *Proceedings of the NATA Advanced Research Workshop on Climate-Ocean Interaction* (1988), M. Schlesinger, Ed., pp. 87–97.

[12] Cox, M. D. A primitive equation, 3-dimensional model of the ocean. Tech. Rep. Tech. Rep. No. 1, GFDL Ocean Group, 1984.

[13] Hoffman, G. R., and Maretis, D. K. *The Dawn of Massively Parallel Processing in Meterology.* Springer-Verlag, Berlin, 1990.

[14] Khokhar, A., Prasanna, V. K., Shaaban, M., and Wang, C.-L. Heterogeneous Supercomputing: Problems and Issues. In *Proceedings of the 1992 Heterogeneous Workshop* (1992), IEEE CS Press.

[15] Levi, B. G. The geometric phase shows up in chemical reactions. *Physics Today 46*, 3 (March 1993), 17–19.

[16] Mechoso, C. R., Farrara, J. D., and Spahr, J. A. Running a climate model in a heterogeneous, distributed computer environment. In *Proceedings of the Third IEEE International Symposium on High Performance and Distributed Computing* (August 1994), pp. 79–84.

[17] Mechoso, C. R., Ma, C.-C., Farrara, J. D., Spahr, J. A., and Moore, R. W. Distribution of a climate model across high-speed networks. In *Proceedings of Supercomputing 1991* (1991), pp. 253–60.

[18] Mechoso, C. R., Ma, C.-C., Farrara, J. D., Spahr, J. A., and Moore, R. W. Parallelization and distribution of a coupled atmosphere-ocean general circulation model. *Monthly Weather Review 121*, 7 (July 1993), 2062–76.

[19] Snyder, L. *Phase Abstractions for Portable and Scalable Parallel Programming.* MIT Press, 1990.

[20] Sunderam, V. S., Geist, G. A., Dongarra, J., and Manchek, R. The pvm concurrent computing system: evolution, experiences, and trends. *Parallel Computing 20*, 4 (April 1994), 531–45.

[21] UNESCO. *UNESCO Technical papers in Marine Science number 36.* UNESCO, Paris, 1981, ch. Tenth report of the joint panel on oceanographic tables and standards.

[22] Wu, M., and Kuppermann, A. Casa quantum chemical reaction dynamics. In *1994 CASA Gigabit Network Testbed Annual Report* (1994).

[23] Wu, Y.-S. M. Personal communication, 1994.

[24] Wu, Y.-S. M., and Kuppermann, A. Prediction of the effect of the geometric phase on product rotational state distributions and integral cross sections. *Chemical Physics Letters 201* (January 1993), 178–86.