Contents

Project Summary

This research proposes a preliminary evaluation of enabling technologies for 100 teraFLOP class computers. The research is broadly classified into the following parts: architectures (single node architecture, parallel architecture), system software (languages, compilers, programming tools and environments), algorithms (design models, performance and scalability), and applications (scaling and realizable performance). Single node architectures for future generation computers will exploit considerable implicit parallelism. Candidate technologies for single node architectures include super-scalar execution, very long instruction words, simultaneous hardware multithreading, and processors-in-memory. We propose a processor architecture based on the VLIW with possible implementations using PIMs. Parallel architectures are based on clusters of these nodes in a hierarchical network. Due to the large number of computational units, algorithm design and scalability issues become critical. Hierarchical decomposition of problems with an emphasis on proper mapping to the underlying architecture is necessary. Whereas problems at higher levels must be loosely coupled, they can be very tightly coupled at lower levels. Alternate algorithm design methodologies and abstract machine models must be constructed. Finally, the scope of applications that can effectively use these architectures needs to be identified. This research will perform a study of each of these aspects of future generation computers. The PIs expect to draw heavily from their experience in the design of scalable parallel algorithms and architectures and from the CEDAR project.

Results from Prior NSF Support for Vipin Kumar and Ahmed Sameh

(i) Highly Parallel Direct Solvers for Sparse Linear Systems, NSF : CCR-9423082 (7/1/95-6/30/98), $191,471. Vipin Kumar is the PI on this grant.

The objective of this project is to develop highly parallel formulations of kernel algorithms in a direct solver, and investigate their applications. The two critical computational elements of direct solvers are: Cholesky (or L/U) factorization, and computations of fill reducing ordering for the matrix. We have recently achieved a breakthrough in developing a highly parallel sparse Cholesky factorization algorithm that substantially improves the state of the art in parallel direct solution of sparse linear systems—both in terms of scalability and overall performance [9]. This algorithm incurs less communication overhead than any previously known parallel formulation of sparse matrix factorization, and therefore, is suitable for workstation clusters that tend to be connected via relatively low-bandwidth and high-latency channels relative to the traditional MPP platforms. The implementation on the Cray T3D delivers up to 20 GFlops on 1024 processors for medium-size structural engineering and linear programming problems. This is the highest performance ever obtained on any supercomputer (vector or parallel) for sparse Cholesky factorization.

Fast and high quality graph partitioning scheme is crucial for developing a scalable parallel direct solvers for sparse linear system so that the step for computing fill-reducing ordering does not become a bottleneck in the overall algorithm, especially given that now scalable and high performance formulation for Cholesky factorization is available. We have developed a high quality and extremely fast scheme for partitioning large irregular graphs. This scheme is one to two orders of magnitude faster than the state-of-the-art multi-level Spectral schemes, and provides partitions that are of substantially better quality quite consistently. Graphs containing half-a-million nodes can be partitioned 256-ways in less than a minute on a scientific workstation . We have also developed a preliminary parallel formulation of this scheme that speeds up the partitioning process even further and is able to find partitions in few seconds. This makes it feasible, for the first time, to dynamically partition graphs in adaptive computations. Serial graph partitioning algorithm (called Metis) has been made available in public domain (over the WWW), and is being used world-wide. Our results show that parallel ordering algorithm based upon our new graph partitioning algorithm takes less time than our highly scalable parallel sparse Cholesky factorization algorithm for all large matrices.

The dissertation of Kumar's PhD student Anshul Gupta (containing results on highly parallel sparse matrix factorization) was selected as one of the five finalists in **The ACM Doctoral Dissertation Competition for 1995**. Papers by Kumar's students Anshul Gupta and George Karypis on Scalable Sparse Matrix Factorization won Outstanding Student Paper Award from Supercomputing 94, and Mannheim SuParCup Prize from the 1995 European Supercomputing conference.

(ii) Acquisition of A Workstation Cluster for Research in High-Performance Computing, Vipin Kumar is a co-PI for this grant (NSG/STI-9414015) which was funded for $550,260 (with matching funds of $484,064 by the University of Minnesota) for the period June 1994 - September 1997.

The goal of this project is to acquire a hierarchical cluster of workstations connected via a switch, and use it to solve large scale problems. The communication latencies in such a platform are much higher than the corresponding latencies in conventional MPPs. They also have a hierarchical structure, as the processors within a workstation are tightly coupled via shared memory. Therefore, it is necessary to develop a toolkit of basic communication operations and a library of commonly used computational building blocks composed of core algorithms specially tailored to these environments. We have considerable expertise in implementing such building blocks on a variety of parallel computers. In particular, we have developed core algorithms for balancing load dynamically among processors for MPPs such as the Cray T3D, nCUBE, CM-200, CM-5, and KSR1. We are in the process of developing efficient parallel formulations of these core algorithms and libraries tailored to a workstation cluster. These libraries can then be used by application programmers to design efficient programs to solve their problems. We are also working on algorithms for solving large sparse linear systems, as well as kernel algorithms used in mining large data bases. The expertise generated in this project will be very valuable for our proposed work, as both require programming on a hierarchical environment.

(iii) Applications over High-Speed Networks: A Pilot Project for the NII, NSF, 1.5 Million (with matching funds of $700,000 by the University of Minnesota), 1995-2000, (Vipin Kumar is co-PI with David Du (PI), Woodward, Yew).

The objective of this proposal is to construct and develop a distributed scalable environment for storage, computation, and visualization over high-speed local and wide area networks to support existing NII applications and to study enabling technologies for NII. Vipin Kumar has been investigating the use of this cluster to solve large scale problems in computational science and in data mining as described in the previous paragraph.

Architecture, Algorithms and Applications for Future Generation Supercomputers

Principal Investigator
V. Kumar and A. Sameh
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

PROJECT DESCRIPTION

1 Introduction

The impetus for the next generation computer systems capable of 100 TeraFLOPS in a 10 year timeframe goes beyond the evolutionary push. Competition in the global marketplace has escalated computer modeling as the primary design and prototyping tool. For many physical phenomena, theoretical models are not well understood. For such problems, simulation is the only alternative. New computational techniques have been developed that promise very accurate modeling. Emerging applications in data intensive computing impose added requirements on machine capabilities. These provide the underlying motivation for a study of enabling and desirable technologies for next generation parallel computers.

From the petaFLOPS architectures workshop at Pasadena, the petaFLOPS forum at Frontiers '95 and the petaFLOPS summer study at Bodega Bay [37, 38], it became apparent that a lack of new architecture models restricts us to architectures that can be defined in terms of conventional concepts. However, the workshops served the very important purpose of consolidating design options and application requirements. These are particularly relevant for the more immediate goal of 100 teraFLOP class machines.

Over the past decade, increases in single node performance have resulted from faster devices and increasing use of implicit parallelism. While device switching speeds are expected to improve further, the gains from it are expected to diminish. Future improvements in speed will hinge on the ability to effectively harness parallelism (both implicit and explicit). It is reasonable to expect that single node performance of workstation class processors will be in the range of 10 gigaFLOPS within 5 - 10 years. This has several implications for parallel computers built using these processors.

At the level of a single node, the gap between peak performance and realizable performance must be bridged. This requires improved compilation techniques, latency hiding mechanisms for memory access and alternate memory structures. Increased single node performance has serious implications for network bandwidth. High network bandwidth requirements coupled with commercial constraints of scalable design suggest a hierarchical interconnection network. Recent advances in symmetric multi-processor design make them ideal candidates for nodes in the hierarchical networks. Programmability constraints make it desirable to explore stronger cache coherence models and alternate programming paradigms and languages. To retain market viability the machine must also support emerging data intensive applications such as parallel databases and data mining. This requires support for parallel/distributed file systems and parallel I/O.

The concurrency requirements of these machines coupled with the hierarchical architecture warrant a rethinking of the design and analysis of algorithms. At the lowest level, strong architecture models allow us to balance load much more aggressively since communication overheads are relatively small. In contrast, at the higher levels, since latency is much higher, the load imbalance must be traded off against communication overheads. We anticipate a very lose coupling between subproblems at higher levels. (For *e.g.*, in a multidisciplinary design optimization problem, different clusters could work on various problems and exchange intermediate results infrequently). Scalability aspects of parallel algorithms must be emphasized due to the large number of processors in such systems. Important applications that need this computing power must to be identified. The ability of these applications to utilize candidate architectures should be carefully analyzed. Where a choice of algorithms exists for a given application, the tradeoffs of using various algorithm-architecture combinations needs to be studied.

This proposal outlines a study of various architecture, system software, algorithm, and application design options to evaluate their suitability for future generation computers.

2 Candidate Single-Node Architectures

The past few years have seen a continued, sometimes even an accelerated improvement in microprocessor speeds. This improvement can be attributed to faster semiconductor technology and to the use of implicit parallelism. The fastest microprocessors currently available operate in the range of 2 - 5 ns clock cycles. It is largely acknowledged that this clock cycle cannot be reduced significantly in the next ten years barring major breakthroughs in device technology. Further improvements in processor speed must come via multiple functional units on a CPU, which requires an aggressive exploitation of implicit parallelism in programs. In contrast to processor speeds, memory speeds have remained largely constant. This has necessitated the use of memory hierarchies, wide memories and threaded architectures to prevent the processor from starving for data. In this section, we explore the space of single node architectures, their advantages and drawbacks.

The design objective of fast processors is to circumvent the memory bottleneck and to optimally utilize available parallelism. Before we discuss individual architecture design points, it is necessary to mention two concepts that transcend the boundaries of these architectures: processors-in-memory (PIM) and multi-threaded architectures. These are not mutually exclusive concepts.

With processor speeds far exceeding memory speeds, it is logical to place the processing elements in the memory itself where a significantly higher bandwidth is available. Such architectures are referred to as **processor-in-memory** (PIM) architectures. PIM architectures have been extensively explored in the EXECUBE [30, 31] project, and its variants have been developed by NEC [47], TI and various other chip designers. There are however several issues that need to be addressed before PIMs can be accepted as an economically viable concept. These relate to the organization of the individual processors, the software support, and device related issues such as packaging and cooling. An orthogonal technique for hiding memory latency is multi-threading. Multi-threaded architectures have been extensively explored since the days of the HEP Denelcor. Threads are a very effective means of expressing concurrency. This concurrency can be used in the parallel as well as serial context for hiding memory latency and scheduling on multiple functional units [41, 44, 43, 42, 45].

Currently, the two most popular architectures for incorporating multiple functional units into microprocessors are superscalar units and very long instruction words. Superscalar processors use multiple pipelines operating in parallel. Almost all current off-the-shelf processors utilize superscalar execution. The key to utilizing superscalar architectures effectively is in identifying instructions that can be scheduled simultaneously. Data dependencies, procedural dependencies and resource conflicts must be resolved for scheduling. In addition, memory bottlenecks must be circumvented to keep the processor busy. Idling resulting from scheduling and memory latency is referred to as vertical and horizontal waste. Multithreading can help in reducing these wastes. By hiding latency, horizontal wastage can be minimized. Vertical waste can be minimized by using simultaneous multithreading. Optimal resource utilization in superscalar processors relies on effective scheduling and hardware support for simultaneous multithreading. One drawback of using a large number of these processors is the number of threads required to account for the vertical waste across all the processors. The scaling properties of these techniques when used in large ensembles are not clear and need to be studied.

Very long instruction word (VLIW) computers pack multiple instructions into a single word. These instructions are fetched and scheduled simultaneously on multiple functional units. The compiler takes the responsibility of detecting the required concurrent instruction sets and inserting NOPS as required. These architectures were explored in the 80s by Multiflow (Trace/200) and Cydrome (Cydra-5) and the Yale ELI project. There has been renewed interest in VLIW machines and some of the forthcoming processors from Intel are based on VLIW architectures. The debate between superscalar and VLIW architectures remains unresolved. We briefly discuss the pros and cons of these architectures.

VLIW software assumes an underlying hardware architecture. Superscalar architectures on the other hand rely on hardware based resource allocation and dependency checking. While this results in added hardware costs (for example, about 5% of the total area on UltraSPARC, a 4-way superscalar processor), it results in portable code. At the same time, it raises questions about the scalability of superscalar architectures since the hardware complexity is quadrilateral in the number of superscalar units. It is widely believed that the number of superscalar units cannot be increased significantly without adversely affecting the cycle time. Furthermore, the hardware parallelism detection in superscalar processors operates within a small window of code. The compiler has a more global view of the program and is therefore able to detect parallelism more aggressively. Combined with loop unrolling and trace scheduling, the degree of parallelism available to VLIW processors is much higher than superscalar processors.

Current PEs in parallel computers such as the Fujitsu VPP330 are capable of over 2 GFLOPS. This performance will be available in commodity microprocessors in the near future. It is widely anticipated that the Intel P7 class of processors based on a VLIW architecture will be capable of delivering in the range of 2 GFLOPS of performance. It seems reasonable that improvements in device technologies and more aggressive use of VLIW parallelism and improved compilation techniques would drive single-node performance to about 10 GFLOPS in a 10 year timeframe. Based on this, the individual processing elements in the proposed architecture are VLIW processors.

3 Candidate Architectures

A variety of factors influence the choice of a network architecture. We outline these motivating factors briefly:

- **Economics**: It is imperative that a project of this magnitude be economically viable. This indicates a scalable design in which bigger machines can be constructed from smaller, higher volume components. SMPs and cluster computing have attracted significant attention and indications are that their popularity will continue to grow as their programming environments and application base grows. This makes them ideal building blocks for larger machines. Larger machines designed using SMPs as processing nodes will benefit from the commodity pricing and software base generated by the low-end market. Developments in LAN technology could provide a cost-effective loosely coupled network for connecting these processing nodes or their clusters.

- **Scalability**: The proposed architecture will have a large number of processors. While low degree networks scale well in terms of hardware, they are not ideal for scalable sustained performance. At the other end of the spectrum, completely connected networks such as crossbars are highly desirable but are very expensive to construct for larger number of processors [32]. Exotic technologies such as free space optical interconnects have not matured enough to warrant commitment in a closed-ended project.

- **Feasibility**: Non-blocking switches are technologically feasible for a small number of processors. Aggregates can be constructed by using these switches in multistage networks. These aggregates can be connected using a high bandwidth network to form clusters. At the top level, many such clusters can be connected using a high bandwidth, high latency network. Hierarchical networks have been used earlier in a variety of machines. Specifically, we expect to draw significantly from our experiences with the **CEDAR** project.

With the above motivations, we propose the following architecture for a 100 TeraFLOP machine that can conceivably be built in a 10 year timeframe.

3.1 Hierarchical Architecture for Scalable Performance

Figure 1 illustrates the proposed candidate architecture. The architecture is hierarchical in nature. At the lowest level, individual PEs are connected to form nodes. These PEs are VLIW processors, each capable of 10 GFLOPS. This is reasonable to expect since 0.8 GFLOPS workstation class processing elements are already available (*e.g.*, the MIPS R10000). We anticipate that stability considerations for numerical methods will require 128 bit math for applications suited to this machine. The processor will therefore have a 128 bit wide word. The interconnection network of a node is a completely connected crossbar network. Individual links in the crossbar network operate at 5 Gbytes/s. The crossbar also supports a 50-100 Gbytes/s channel out of the node. Each crossbar also has a link going to a disk subsystem. This allows fast access from the nodes to the data, which is particularly important for data-intensive applications. Crossbars operating at this speed will conceivably evolve naturally from the current state of the art. In the 10 year timeframe, we expect these 100-200 GFLOPS nodes to be the mini-super computers. These will have a commodity status, much like todays bus-based SMPs (*e.g.*, PowerChallenge arrays at the current state-of-the-art).

The primary motivation for this design is to develop a powerful compute engine with a significant commercial market. This places particular emphasis on the programmability of the node. The programmer must not have to deal with the intricacies of data placement for general applications. This requires a high bandwidth, low latency interconnect to the memory. A non-blocking crossbar network is an ideal network for small numbers of processors. All memory in the system is accessible to the processors only through the crossbar. This provides a measure of location independence at the cost of added crossbar latency for memory accesses. However, since switching and computing speeds are growing much faster than memory speeds, we do not expect a significant overhead when the crossbar latency is stacked on memory latency.

The node architecture supports a sequential consistency model. The crossbar also supports links that provide connectivity to the rest of the network. An external processor gets a homogeneous view of the memory. Trends in SMP design have already started exploring this design. Examples of machines (or machine nodes) based on a crossbar include the Cray T90, the Convex VPP, AlphaServer 2100, RS/6000 SMP servers, and the UltraSparc SMP.

The node architecture presented above is not expected to scale up to more than 20 processors without

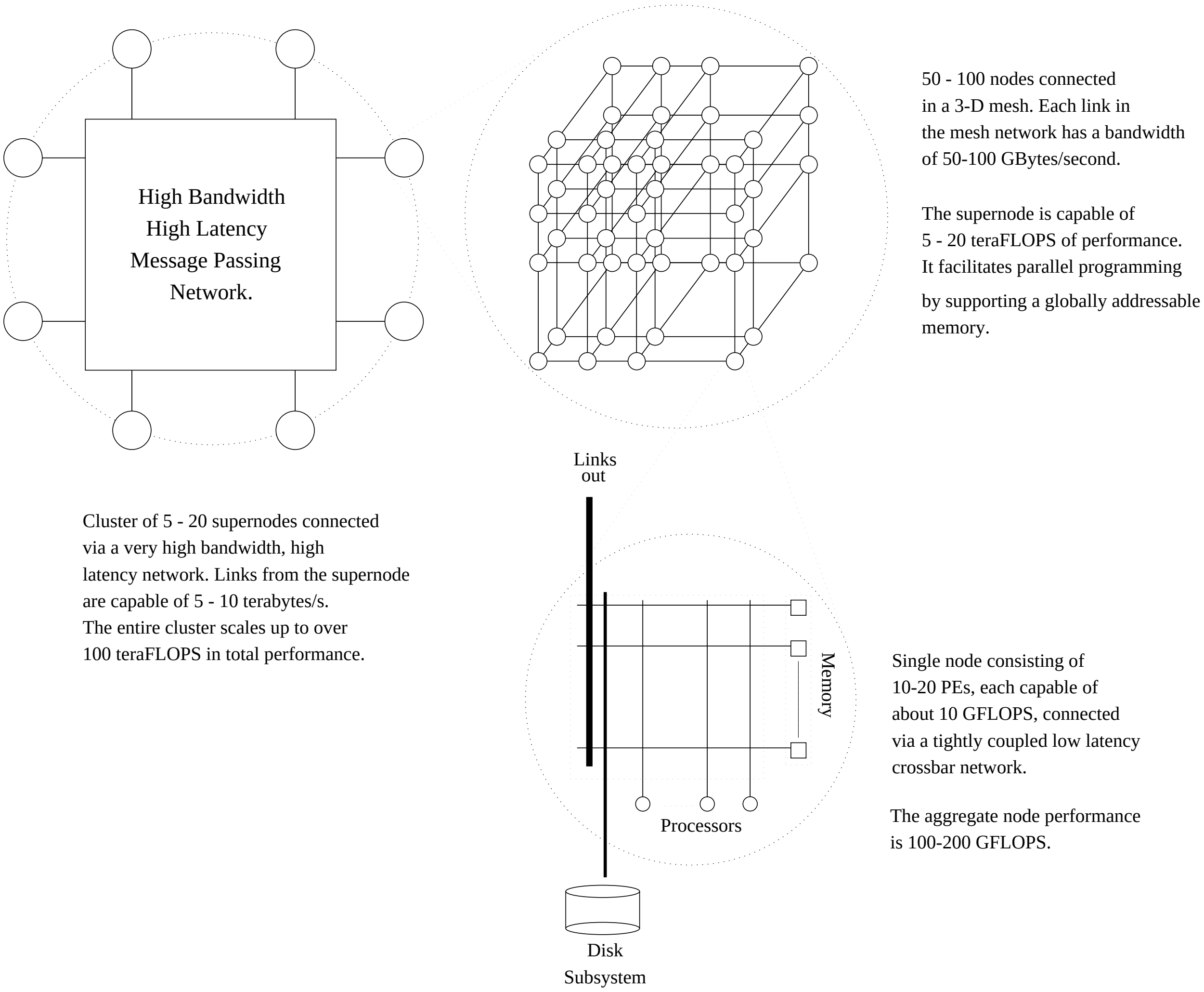

Figure 1: Hierarchical structure of the proposed architecture: PEs connected in a crossbar to form a node, nodes connected in a 3-D toroidal mesh to form a supernode, cluster of supernodes connected via a high bandwidth, high latench network.

compromising the network performance. The reason is that paths between memory and PEs will have to be very wide (4 - 16 128 bit words) to ensure good performance from the 10 GFLOPS PE. We therefore must resort to a hierarchical network with a sparser network at higher levels. At the next level of the hierarchy, 50-100 nodes are connected via a 3-D toroidal network to form a supernode capable of 5-20 teraFLOPS of performance. Individual network links in the mesh operate at 50-100 Gbytes per second. The supernode also supports an embedded network of routers which provide fast access to other supernodes via the high bandwidth switch. Specifically, a group of four nodes shares a single network router. We expect these supernodes to be the counterparts of current day MPPs. The supernode supports a globally addressable memory. A small number (5 - 20) of these supernodes are connected into a cluster using a very high bandwidth, high latency interconnection network.

The proposed architecture is a very scalable design. The only component specific to the 100 teraFLOPS machine is the high bandwidth, high latency network. We expect all other components to have commodity status and therefore their development costs will be supported at least in part by the market. Even the high bandwidth network for the cluster could be made using off-the-shelf building blocks from current and future generation LANs such as those offered by Myricom and AT&T. Higher bandwidth is obtained by wider channels. The use of optic fiber is also a possibility. In either case, the startup latency will be relatively large. Therefore, at the cluster level, we will only exploit coarse-grain parallelism. In addition to the hierarchical architecture, we envision a hierarchical software support.

4 Programming Environment

The hierarchical nature of the proposed architecture necessitates programming tools that allow a hierarchical programming model. Software that supports the programming model that is better suited for a given level of the hierarchy must be provided. This programming environment allows the programmer to concentrate on developing software for a given level of the hierarchy without the burden of architectural details of the entire hierarchy. Various levels of the hierarchical model are based on well understood programming paradigms.

4.1 Support for Individual PEs

The gap between peak and sustained performance of current RISC processors underlines the need for advances in compiler technology. Code optimization for memory hierarchy is a significant problem. A variety of promising technologies will be explored as a part of this research. Realistic machine models will be developed for VLIW compilation. A key observation that can be made from optimized codes today is the fact that efficient serial algorithms are based on efficient parallel algorithms [8]. For example, most matrix algorithms are based on blocking the matrix, much the same as parallel algorithms for the same problem [11]. This is expected, since the core objective of localizing remote accesses is common to both serial and parallel algorithms. This lends hope to the development of a machine model for the development of fast serial algorithms. Efficient compilers can then be built around this abstract machine model.

Multi-threading has emerged as an important tool for hiding memory latency and minimizing idling in the PE. Although VLIW compilers detect instruction level threads implicitly, their scalability in the multiplicity of functional units has not been demonstrated. Further, the role of multithreading support for VLIW architectures needs to be investigated. Nevertheless, we expect these technologies to mature significantly with the advent of VLIW machines into the mass market (*e.g.*, the P7 microprocessor).

4.2 Support for Nodes

The **nodes** of the proposed architecture are similar to today's cache coherent SMPs and PVPs, which already have a rich set of well understood programming tools. Programs can be developed using automatic parallelization tools, serial programming languages augmented with explicit parallelization directives, or using explicit shared memory programming languages.

4.3 Support for Supernodes

The supernodes are distributed memory, global addressable, cache incoherent parallel computers. Their architecture resembles that of existing MPP systems, namely Cray T3D and T3E. Even though the software support for these systems is not as versatile as that of the **nodes**, existing programming models can be used to efficiently program the supernodes.

This level of the hierarchy will be programmed using the SPMD programming model. Data parallel languages similar to HPF and HPC++ will be used to develop efficient parallel programs. These languages will provide explicit data placement and concurrency support, but will allow an implicit expression of the computation, in form of a library call. These library calls will essentially be parallel programs to be executed by the **nodes** of the architecture.

4.4 Support for the Cluster

The cluster level view of the proposed architecture is that of a distributed memory parallel computer. The best suited programming model for such architecture is explicit message passing. Programming tools will be provided, such as MPI, for communication among the supernodes of the cluster. These message passing libraries will express data transfer between supernodes, as if these supernodes where single processors. As it was the case of the programming tools at the supernode level, computation will again be expressed in terms of library calls, which now correspond to parallel programs for the **supernodes**.

4.5 Debugging and Performance Evaluation Tools

Developing debugging and performance evaluation tools is critical for significantly reducing software development time and improving the performance of the parallel programs developed for the proposed architecture. These tools will be provided for each level of the hierarchy, and similarly to the provided programming languages, they will hide the details of the entire hierarchy from the programmer. For example, the debugging and performance evaluation tools that will be provided for the **supernodes**, will provide the programmer a view of the system as if each **node** was a single processor; thus alleviating the complexity of programming the lower level of the hierarchy.

5 Algorithm Design and Analysis

One of the main advantages of the proposed hierarchical architecture is that it allows us to develop a scalable architecture that captures the best features of each level of the hierarchy. As a result, algorithms designed for this architecture must also take advantage of the features of each hierarchy level.

This new class of algorithms (**hierarchical algorithms**) start as regular message passing algorithms, performing computation and communication. Each computation phase in turn is another parallel algorithm that may be expressed in a data parallel style, consisting of another set of interleaved phases that perform computation and communication. Finally, the computation at this level is expressed as a pure shared memory parallel program that takes advantage of the tightly coupled processors at each **node** of the architecture. Thus, the hierarchical algorithm essentially has three different levels. The algorithm at each level will be chosen that best matches the architecture at that level. For some problems, these algorithms will be identical for all levels. However, for some problems (*e.g.*, the classification problem discussed in Section 6), a different algorithm may perform better for a tightly coupled architecture than for a distributed memory system.

Applications that exhibit either data or task parallelism can take advantage of this algorithm design methodology. Developing hierarchical algorithms does not require significant additional investment, since good parallel algorithms for each level of the hierarchy are already known for many problems. Furthermore, many of these will be needed for good performance even for the single PE due to deep memory hierarchies.

Hierarchical algorithms, besides being easy to develop, due to their nature. they also allow the development of reusable libraries for each level of the architectural hierarchy. Consequently, many diverse applications can benefit from the existence of these highly parallel and high performance library routines that address the common computational requirements (*e.g.*, sparse numerical methods) common to many of these applications.

5.1 Models for Algorithm Design

The absence of an abstract machine model for parallel computers is one of the basic reasons for the relative difficulty in designing parallel algorithms. In the recent past, several abstract models such as BSP, LogP, and YPRAM have been proposed. The scope of these models in the space of architectures and algorithms is limited. None of these models are designed to handle hierarchical architecture models. We propose to develop abstract models for hierarchical machines [8].

5.2 Metrics for Algorithm Analysis

The large number of PEs in the system imposes significant requirements on the scalability of the algorithm-architecture combination [35, 12]. A significant part of applied and analytical parallel computing deals with performance properties up to hundreds of processors. This must be extended to the range of processors in the proposed architecture. Clearly, as the number of processors is increased, the problem size must be increased to maintain efficiency [3, 32]. This implies a certain growth rate for the total memory in the system. Furthermore, the solution time may also increase with increasing number of processors. Since we cannot expect to allow arbitrary solution times, this must be factored into the scalability analysis. Therefore, scalability metrics must incorporate available memory, solution time, performance, and concurrency inherent in the application. A number of scalability metrics have been proposed and studied:

Fixed Efficiency Metrics It is known that as the number of processors used to solve a given problem is increased, the overall efficiency reduces. However, for a class of algorithms and architectures, this efficiency can be propped up by compensating an increase in the number of processors with an increase in the problem size. We refer to such systems as *efficiency-scalable*. The rate at which the problem size must be increased with increasing number of processors to maintain the efficiency constant is an excellent measure of the scalability of the system [35, 3]. We refer to this rate of increase in problem size as the *isoefficiency* of the parallel system. We have demonstrated the utility of the isoefficiency metric in the context of a variety of problems [32, 15, 11, 14, 33, 29, 27]. We have also established the relationship of the isoefficiency metric to other fixed time and fixed memory metrics.

Memory Scaling Metrics The rate of growth of problem size may also be constrained by the amount of memory in the parallel computer. These metrics are critical for machines of the proposed size and the scope of targeted applications. Memory capacity requirements vary significantly for different applications. A large and general class of simulation-oriented applications exhibit memory scaling requirements less than or equal to the 3/4 the computational requirement. The corresponding memory requirements of these machines are in the range of a few terabytes. Data-intensive applications on the other hand require large amounts of memory, in excess of 100 terabytes for the proposed system. In contrast, time-critical applications such as active control systems require significantly less memory. Often, a terabyte or less of memory will suffice for these applications.

The memory scaling characteristics of the application determine the overall memory available in the system. The problem size is increased at the fastest rate allowed by the available memory. The resulting performance characteristics determine the scalability of the system. These metrics have been investigated by Worley [46], Gustafson [17] and Sun and Ni [40].

Fixed Time Metrics In many situations, the rate of growth of problem size is dictated by the time available on the machine to solve the problem. In these cases, the problem size is increased with number of processors in such a way that the parallel runtime remains constant. The scalability issues for such problems have been explored by Worley [46], Gustafson [17], and Sun and Ni [40].

We have studied these and other scalability metrics extensively [35, 12, 3]. We will evaluate the suitability of these metrics in the context of the proposed architecture. If necessary, we will develop new scalability metrics which will incorporate the desirable features of all these metrics and use them to analyze candidate applications.

6 Candidate Applications and Scaling Properties

6.1 Sparse Matrix Methods in Large Scale Numerical Simulations

Numerical simulations are essential in modeling physical phenomena, manufacturing and design, and computational medicine. Being able to simulate problems using large three-dimensional unstructured meshes (10-100 million elements) will allow the modeling of complex objects, such as entire airplane/car in aerospace and automotive industries, or various parts of the human body in medicine.

However, such large scale simulations involve meshes that are highly irregular and inhomogeneous, resulting in ill-conditioned systems. Consequently, traditional global-preconditioning methods such as those based on incomplete LU factorization tend to produce very poor results. One way of providing robust preconditioners is to combine the strengths of iterative methods (smaller memory requirement, faster solution time), with the power of direct methods (robustness); thus creating hierarchical preconditioning methods. We have studied various aspects of sparse linear system solvers on conventional parallel computers [9, 15, 28, 26, 22, 19, 23, 24, 20].

One such preconditioning technique can be based on the algorithm developed by Hu, Lou and Sameh [18] that uses a domain decomposition into overlapping domains to produce robust preconditioners that significantly outperform preconditioners based on incomplete LU factorization. This algorithm, decomposes the domain into k overlapping domains. Let M be the matrix that corresponds to the overlap. The algorithm iterates over M (which is never actually formed) in order to solve the iterative system. The matrix-vector multiplication of M with a vector x requires the solution of a system for each subdomain.

On the proposed architecture with P_1 supernodes each containing P_2 nodes, and each node corresponding to P_3 processors, such a hierarchical preconditioner can be applied as follows: The mesh with N elements is decomposed into P_1 overlapping domain [20, 23, 24] each of size N/P_1, and let M_1 be the matrix that corresponds to the overlap. Since iterating on M_1 requires the solution of systems involving the subdomains, each such system is solved in a similar hierarchical fashion as follows. Each subdomain to be solved by each supernode, is recursively decomposed into P_2 subdomains [24], each of size $N/(P_1 * P_2)$.

At this point, each subdomain is solved using direct methods on each node. This computation requires $O(N^2/(P_1 * P_2))$ computation, and the resulting triangular systems can be solved in $O(N^{4/3}/(P_1 * P_2))$ operations. Since each node achieves 100-200GFlops, a linear system of size of size 10^5 can be solved in around a second. Thus, linear systems of size 10^8 to 10^9 can be solved in a few minutes. This will allow the simulation of problems whose meshes contains over 100 million elements with multiple degrees of freedom.

6.2 Particle Dynamics and Boundary Element Methods

A number of problems are characterized by dense matrices. These include problems in particle dynamics (simulations in astrophysics, molecular dynamics, material and device properties, particle physics, vorticity methods for fluid flows, and protein synthesis to name a few). The underlying computational substrate of these methods is a particle simulation method based on hierarchical treecodes or lattice based methods. Hierarchical treecodes such as Barnes-Hut and Fast Multipole Method (FMM) speed up the dense problem by evaluating the interactions approximately. In doing so, they reduce the complexity of many dense problems from $O(n^2)$ to $O(n)$ or $O(n \log n)$. Even with this reduced complexity, the computational requirements of these methods require very high computational speeds.

As an example, consider a problem in molecular dynamics. The resonating frequency of a hydrogen atom is in the range of a femto-second. Therefore, a single time-step in the simulation must be 10^{-15} seconds. Meaningful simulations require at least a microsecond of simulated time. This corresponds to a million time-steps. If we want this simulation to run in about 15 - 20 minutes of dedicated time, each time-step would need to be computed in about a millisecond. On a 100 teraFLOPS machine, this corresponds to a simulation of about 10^7 particles. Simulations of this magnitude are very useful for understanding various physical phenomena and in applications such as protein synthesis, drug design etc. We have studied parallel formulations of treecodes for particle dynamics on conventional MPPs extensively [7, 4, 6] and have demonstrated excellent scalability and performance.

Many other applications of hierarchical methods have similar computational requirements. These applications include a variety of inverse problems arising in the solution of integral equations. The dense linear systems in these methods are solved using iterative solvers. Each matrix-vector product in the iterative solver can be computed as a single iteration of a treecode. These techniques find very important applications in electromagnetics, acoustics, and imaging. A very important application inverse problems is in determining the radar cross-section of aircraft. At high wave numbers, the surface discretization of the aircraft must be very fine. This corresponds to a large number of variables. Solving for discretizations corresponding to a million boundary elements are desirable. The convergence of these solvers is accelerated using scalable preconditioning techniques. We have used our parallel formulations of hierarchical methods for developing dense solvers and preconditioners [5, 6]. We have demonstrated the excellent performance and scalability of these methods.

Parallel formulations of hierarchical methods map naturally to the hierarchical architecture proposed. The top level nodes of the tree are duplicated across the processors. The resulting problem reduces to projecting the influence of individual domains assigned to processors on other domains. The computation involved with individual domains is significantly higher than the resulting communication. We have demonstrated that parallel formulations can be coupled with communication aggregation techniques to hide high communication latency. In addition, we have explored scalable preconditioners that can be used for the inverse problem.

Our extensive experiences with treecodes leads us to believe that our parallel formulation will scale up to utilize the proposed architecture efficiently. As a part of this research, we will study the performance of hierarchical methods analytically in the context of particle simulation problems and inverse problems for the proposed architecture.

6.3 Emerging Large-Scale Applications: Data Mining

Knowledge discovery (data mining) from huge databases is a very important and rapidly growing application. It is anticipated that by the year 1997, the turnover of the data-mining software industry itself (not including associated hardware) will exceed five billion dollars. This is more than the entire supercomputing industry currently.

One of the important problems in data mining [39] is the classification-rule learning. Classification-rule learning involves finding rules or decision trees that partition the given data into predefined classes. For any realistic problem domain of the classification-rule learning, the set of possible decision trees is too large to be searched exhaustively. All of the existing algorithms, like C4.5 [36] and CDP [1], use local heuristics to construct the decision trees. The computational complexity of these algorithms ranges from $O(n \log n)$ with discrete data attributes only, to $O(n \log^2 n)$ with continuous data attributes. We have improved the performance of these algorithms and explored their parallel formulations [2].

We can map the classification-learning algorithms based on ID3, like C4.5, on the proposed architecture nicely. The C4.5 algorithm generates a decision tree for the given training data set by recursive partitioning the data. On the proposed architecture the n element data-set can be assigned to P supernodes by assigning subsets of n/P elements to each. The decision tree can be built breadth-first manner in all of the supernodes by exchanging the frequency information of the local data. As soon as the number of the leaf nodes on the frontier reaches the number of supernodes P, the training data is shuffled such that each supernode has a subset of the data corresponding to one of the leaf nodes of the decision tree. Since the cluster is connected via a very high bandwidth network, this shuffling can be done very efficiently.

After the data is shuffled, each supernode has a totally independent task of the building a subtree of the leaf node it represents. No communication across supernodes is required. Each supernode builds a subtree based on the local data. At this level, static and dynamic load balancing schemes can be used to maximize the computation efficiency. The data can be further partitioned along the planes of the 3-D mesh such that each plane represents a leaf node of the subtree. This hierarchically structured algorithm is ideally suited for the proposed architecture.

We anticipate that such a parallel formulation is capable of yielding excellent scalability and performance. We can use it to solve classification problems for databases with 10^{12} records or more. Such databases are routinely gathered from commercial sources and valuable information can be mined from them. For 10^{12} records, the operation count of an $O(n \log^2 n)$ algorithm with a constant of 100 is 1.6 petaOps. A problem of this size can be solved in approximately half an hour on a 100 teraOps machine. The impact of this capability is tremendous. Very often, the discovery process in data mining has to be guided using visualization tools [2]. In such cases, a short turnaround from the mining process is critical.

7 Summary of Proposed Research

In this proposal, we have outlined a hierarchical architecture for machines capable of over 100 teraOpS in a 10 year timeframe. The motivating factors for the design are technological feasibility and economic viability. The envisioned architecture can be built largely from commodity components. The development costs of the machine will therefore be shared by the market. We feel that this is critical for an effort of the required magnitude.

To obtain sustained performance from the machine, we propose a heterogeneous programming environment for the machine. The programming environment optimally uses the power of the hierarchy. Programming models for the stronger machine models existing at the lower levels are tuned for ease of programming. Higher levels of the hierarchy place progressively greater emphasis on locality of data reference.

The envisioned machine architecture requires new algorithm design methodologies. We propose to develop hierarchical parallel algorithms and scalability metrics for evaluating such algorithms. We identify three important application areas: large scale numerical simulations, problems in particle dynamics and boundary element methods, and emerging large-scale applications such as data-mining. We briefly outline the process of hierarchical algorithm design for each of these application areas. We propose to study the performance of these applications for the outlined architecture.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engg.*, 5(6):914–925, December 1993.

[2] M. Ganesh, E.H. Han, V. Kumar, S. Shekhar, and J. Srivastava. Visual data mining: Framework and algorithm development. Technical Report TR-96-021, Department of Computer Science, University of Minnesota, Minneapolis, 1996.

[3] Ananth Grama, Anshul Gupta, and Vipin Kumar. Isoefficiency function: A scalability metric for parallel algorithms and architectures. *IEEE Parallel and Distributed Technology, Special Issue on Parallel and Distributed Systems: From Theory to Practice*, 1 (3):12–21, 1993. Also available as Technical Report TR-93-24, Department of Computer Science, University of Minnesota and from anonymous ftp site ftp.cs.umn.edu, file users/kumar/isoeff-tutorial.ps.

[4] Ananth Grama, Vipin Kumar, and Ahmad Sameh. Scalable parallel formulations of the barnes-hut method for n-body simulations. In *Supercomputing '94 Proceedings*, 1994.

[5] Ananth Grama, Vipin Kumar, and Ahmad Sameh. Parallel iterative solvers and preconditioners using approximate hierarchical methods. In *Proceedings of the Copper Mountain Conference on Iterative Methods*, 1996.

[6] Ananth Grama, Vipin Kumar, and Ahmad Sameh. Parallel matrix-vector product using hierarchical methods. In *Proceedings of Supercomputing '95*, San Diego, CA, 1995.

[7] Ananth Grama, Vipin Kumar, and Ahmed Sameh. On n-body simulations using message passing parallel computers. In *Proceedings of the SIAM Conference on Parallel Processing*, San Francisco, 1995.

[8] Ananth Y. Grama, V. Kumar, S. Ranka, and V. Singh. On architecture independent design and analysis of parallel programs. Technical report, University of Minnesota, Minneapolis, 55455, 1996.

[9] Anshul Gupta, George Karypis, and Vipin Kumar. Highly scalable parallel algorithms for sparse matrix factorization. Technical Report 94-63, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1994. Submitted for publication in *IEEE Transactions on Parallel and Distributed Computing*. Available on WWW at URL http://www.cs.umn.edu/~karypis/papers/sparse-cholesky.ps.

[10] Anshul Gupta, George Karypis, and Vipin Kumar. A highly scalable parallel algorithm for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems (Submitted)*, 1995. A short version of this paper won the Outstanding Student Paper Award from the Supercomputing 94 conference.

[11] Anshul Gupta and Vipin Kumar. The scalability of Matrix Multiplication Algorithms on parallel computers. Technical Report TR 91-54, Computer Science Department, University of Minnesota, Minneapolis, MN, 1991. A short version appears in *Proceedings of International Conference on Parallel Processing*, pages III-115–III-119, 1993. Available via anonymous ftp from ftp.cs.umn.edu at users/kumar/matrix.ps.

[12] Anshul Gupta and Vipin Kumar. Performance properties of large scale parallel systems. *Journal of Parallel and Distributed Computing* (special issue on supercomputer performance), 19(3), November 1993. Also available as Technical Report 92-32, Department of Computer Science, University of Minnesota, Minneapolis, MN. Available via anonymous ftp from ftp.cs.umn.edu at users/kumar/performance.ps.

[13] Anshul Gupta and Vipin Kumar. A scalable parallel algorithm for sparse matrix factorization. Technical Report 94-19, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1994. A shorter version appears in Supercomputing '94. TR available in *users/kumar/sparse-cholesky.ps* at anonymous FTP site *ftp.cs.umn.edu*.

[14] Anshul Gupta and Vipin Kumar. The scalability of FFT on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):922–932, August 1993. Available as a Technical Report TR 90-53, Computer Science Department, University of Minnesota. Available via anonymous ftp from ftp.cs.umn.edu at users/kumar/fft.ps.

[15] Anshul Gupta, Vipin Kumar, and A. H. Sameh. Performance and scalability of preconditioned conjugate gradient methods on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 1995 (To Appear). Also available as Technical Report TR 92-64, Department of Computer Science, University of Minnesota, Minneapolis, MN. A short version appears in *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 664–674, 1993.

[16] Anshul Gupta, Vipin Kumar, and Ahmed Sameh. Performance and scalability of preconditioned conjugate gradient methods on parallel computers. Technical Report TR 92-64, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1992. A short version appears in the *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 664–674, 1993. Available via anonymous ftp from ftp.cs.umn.edu at users/kumar/conjugate-gradient.ps.

[17] John L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[18] Dan Hu, Felix Lou, and Ahmed Sameh. Parallel iterative solvers for almost block-diagonal linear systems. Technical Report CS-96-000, Department of Computer Science, University of Minnesota, 1996.

[19] G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. Technical Report TR 95-037, Department of Computer Science, University of Minnesota, 1995. Also available on WWW at URL http://www.cs.umn.edu/~karypis/papers/mlevel_analysis.ps.

[20] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995. Also available on WWW at URL http://www.cs.umn.edu/~karypis/papers/mlevel_serial.ps. A short version appears in Intl. Conf. on Parallel Processing 1995.

[21] G. Karypis and V. Kumar. Metis: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota, 1995. Available on the WWW at URL *http://www.cs.umn.edu/~karypis/metis/metis.html*.

[22] G. Karypis and V. Kumar. Multilevel graph partition and sparse matrix ordering. In *Intl. Conf. on Parallel Processing*, 1995. Available on WWW at URL http://www.cs.umn.edu/~karypis/papers/mlevel_serial.ps.

[23] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report TR 95-064, Department of Computer Science, University of Minnesota, 1995. Also available on WWW at URL http://www.cs.umn.edu/~karypis/papers/mlevel_kway.ps.

[24] G. Karypis and V. Kumar. Parallel multilevel graph partitioning. Technical Report TR 95-036, Department of Computer Science, University of Minnesota, 1995. Also available on WWW at URL http://www.cs.umn.edu/~karypis/papers/mlevel_parallel.ps.

[25] George Karypis, Anshul Gupta, and Vipin Kumar. Ordering and load balancing for parallel factorization of sparse matrices. Technical Report (in preparation), Department of Computer Science, University of Minnesota, Minneapolis, MN, 1994.

[26] George Karypis, Anshul Gupta, and Vipin Kumar. A highly parallel interior point algorithm. In *Proceedings of the seventh SIAM conference on Parallel Processing for Scientific Computing*, pages 110–112, 1995.

[27] George Karypis and Vipin Kumar. Efficient parallel mappings of a dynamic programming algorithm: A su mmary of results. In *7th International Parallel Processing Symposium*, pages 563–568, 1993.

[28] George Karypis and Vipin Kumar. Fast sparse Cholesky factorization on scalable parallel computers. Technical report, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1994. A short version appears in the *Eighth Symposium on the Frontiers of Massively Parallel Computation*, 1995. Available on WWW at URL http://www.cs.umn.edu/~karypis/papers/frontiers95.ps.

[29] George Karypis and Vipin Kumar. Unstructured Tree Search on SIMD Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, 5(10):1057–1072, October 1994. available as Technical Report TR 92-21, Computer Science Department, Un iversity of Minnesota.

[30] Peter M. Kogge. EXECUBE - a new architecture for scalable mpps. In *International Conference on Parallel Processing*, pages I77–I84, August 1994.

[31] Peter M. Kogge, Toshio Sunaga, Hisatada Miyataka, Koji Kitamura, and Eric Retter. Combined DRAM and logic for massively parallel systems. In *Conference on Advanced Research in VLSI*, pages 4–16, Chapel Hill, NC, March 1995.

[32] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing: Algorithm Design and Analysis.* Benjamin Cummings/ Addison Wesley, Redwod City, 1994.

[33] Vipin Kumar, Ananth Grama, and V. Nageshwara Rao. Scalable load balancing techniques for parallel computers. *Journal of Parallel and Distributed Computing*, 22(1):60–79, July 1994. Also available as Technical Report 91-55 (November 1991), Department of Computer Science, University of Minnesota, Minneapolis, MN. Available via anonymous ftp from ftp.cs.umn.edu at users/kumar/lb-MIMD.ps.

[34] Vipin Kumar and Anshul Gupta. Analyzing the scalability of parallel algorithms and architectures: A survey. In *Proceedings of the 1991 International Conference on Supercomputing*, 1991. Also appears in September 1994 issue of JPDC. Available via anonymous ftp from ftp.cs.umn.edu at users/kumar/survey-scalability.ps.

[35] Vipin Kumar and Anshul Gupta. Analyzing scalability of parallel algorithms and architectures. *Journal of Parallel and Distributed Computing (special issue on scalability)*, 22(3):379–391, September 1994. A short version of the paper appears in the Proceedings of the 1991 International Conference on Supercomputing. Available via anonymous ftp from ftp.cs.umn.edu at users/kumar/survey-scalability.ps.

[36] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[37] Thomas Sterling and Michael MacDonald. The proceedings of the petaflops frontier workshop. Technical Report TR-95-161, Center of Excellence in Space Data and Information Science, Goddard Space Flight Center, Greenbelt, MD, 1995.

[38] Thomas Sterling, Paul Messina, and Paul Smith. *Enabling Technologies for Petaflops Computing.* MIT Press, Cambridge, MA, 1995.

[39] M. Stonebraker, R. Agrawal, U. Dayal, E. J. Neuhold, and A. Reuter. Dbms research at a crossroads: The vienna update. In *Proc. of the 19th VLDB Conference*, pages 688–692, Dublin, Ireland, 1993.

[40] Xian-He Sun and L. M. Ni. Another view of parallel speedup. In *Supercomputing '90 Proceedings*, pages 324–333, 1990.

[41] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In *In 23rd Annual International Symposium on Computer Architecture*, May, 1996.

[42] D.M. Tullsen and S.J. Eggers. Limitations of cache prefetching on a bus-based multiprocessor. In *In 20th Annual International Symposium on Computer Architecture*, pages 278–288, 1993.

[43] D.M. Tullsen and S.J. Eggers. Effective cache prefetching on bus-based multiprocessors. *ACM Transactions on Computer Systems*, pages 57–88, February, 1995.

[44] D.M. Tullsen, S.J. Eggers, and H.M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *In 22nd Annual International Symposium on Computer Architecture*, June, 1995.

[45] D.M. Tullsen and M.D. Ercegovac. Design and vlsi implementation of an online algorithm. In *Real Time Signal Processing IX*, August, 1988.

[46] Patrick H. Worley. The effect of time constraints on scaled speedup. *SIAM Journal on Scientific and Statistical Computing*, 11(5):838–858, 1990.

[47] Nobuyuki Yamashita, Tohru Kimura, Yoshihiro Fujita, Yoshiharu Aimoto, Takashi Manaba, Shin'ichiro Okazaki, Kazuyuki Nakamura, and Masakazu Yamashina. A 3.84GIPS integrated memory array processor lsi with 64 processing elements and 2mb SRAM. In *International Solid-State Circuits Conference*, pages 260–261, San Francisco, February 1994. NEC.

VIPIN KUMAR
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

Education

Ph.D. in Computer Science from University of Maryland, 1982.

Professional Experience

Professor, University of Minnesota, since August 1995.

Associate Professor, University of Minnesota, 1989-95.

Assistant Professor, University of Texas at Austin, 1983-89.

Research Publications

FIVE PUBLICATIONS MOST CLOSELY RELATED TO PROPOSED PROJECT

1. Anshul Gupta, George Karypis and Vipin Kumar, A Highly Scalable Parallel Algorithm for Sparse Matrix Factorization, IEEE Transactions on Parallel and Distributed Systems (submitted) 1995. A short version of this paper won the **Outstanding Student Paper Award from the Supercomputing 94 conference** and the **SuParCup prize from the European Supercomputing Conference**.
2. Anshul Gupta, Vipin Kumar, and Ahmed Sameh, Performance and Scalability of Preconditioned Conjugate Gradient Methods on the CM5, IEEE Transactions on Parallel and Distributed Systems Volume 6, Number 5, pp. 455-469, May 1995.
3. "Analyzing the Scalability of Parallel Algorithms and Architectures: A Survey", Vipin Kumar and Anshul Gupta, *Journal of Parallel and Distributed Computing*, 1993.
4. V. Kumar, Ananth Grama, and V.N. Rao, "Scalable Load Balancing Techniques for Parallel Computers" in *Journal of Parallel and Distributed Computing*, Volume 22, Number 1, pages 60-79, July 1994.
5. "Isoefficiency Function: A Scalability Metric for Parallel Algorithms and Architectures", Ananth Grama, Anshul Gupta, and Vipin Kumar, *IEEE Parallel and Distributed Technology, Special Issue on Parallel and Distributed Systems: From Theory to Practice*, August 1993.

FIVE OTHER SIGNIFICANT PUBLICATIONS

1. Ananth Grama, Vipin Kumar and Ahmed Sameh, "Parallel Iterative Solvers and Preconditioners Using Approximate Hierarchical Methods", proceedings of the Copper Mountain Conference on Iterative Methods, April 1996, Copper Mountain, CO.
2. George Karypis and Vipin Kumar, Parallel Multi-level Graph Partitioning. Proceedings of the International Parallel Processing Symposium, April 1996.
3. "Performance Properties of Large Scale Parallel Systems", Anshul Gupta and Vipin Kumar, *Journal of Parallel and Distributed Computing*, 1993.

4. Ananth Y. Grama, Vipin Kumar and Ahmed Sameh, Parallel Matrix-Vector Product Using Approximate Hierarchical Methods Proceedings of Supercomputing'95, December 1995, San Diego.
5. "Unstructured Tree Search on SIMD Parallel Computers", George Karypis and Vipin Kumar, *IEEE Transactions on Parallel and Distributed Systems*, Volume 5, Number 10, pp. 1057-1072, October 1994.

Books

1. Vipin Kumar, Ananth Grama, Anshul Gupta and George Karypis, "Introduction to Parallel Computing: Algorithm Design and Analysis", Benjamin-Cummings, 1994.
2. Parallel Processing for Artificial Intelligence, Volume 1, edited by Laveen Kanal, Vipin Kumar, Hiroaki Kitano and Christian B. Suttner, North-Holland, June 1994.
3. Parallel Processing for Artificial Intelligence, Volume 2, edited by Hiroaki Kitano, Vipin Kumar, and Christian B. Suttner, North-Holland, July 1994.
4. Parallel Algorithms for Machine Intelligence and Vision, edited by V. Kumar, P.S. Gopalkrishnan, and L. Kanal, Springer-Verlag, March 1990.

Honors

1. Plenary Speaker, 16th IFIP Conference on System Modeling and Optimization, July 1993, France.
2. Honorable Mention, 1989 Gordon Bell Prize Competition.
3. Dissertations of two PhD students were among five finalists in the ACM Doctoral Dissertation Award competition (Yow-Jian Lin (1988) for his work on automatic parallel execution of logic programs and Anshul Gupta (1995) for his work on highly scalable sparse Cholesky factorization.

Awards

Graduate Student Research Excellence Award from the Maryland chapter of Sigma-Xi, the Scientific Research Society, 1982.

A.C.M. Samuel Alexander Fellowship Award from the Washington D.C. chapter of the Association of Computing Machinery, 1981.

Professional Activities

1. Editorial Board, **IEEE Transactions on Knowledge and Data Engineering**, 1993 – present.
2. Editorial Board, **Journal of Parallel and Distributed Computing**, 1994 – current.
3. Editorial Board, **IEEE Parallel and Distributed Technology**, 1995 – current.
4. Editorial Board, **Parallel Computing**, 1996 – present.
5. Guest Editor, **Journal of Parallel and Distributed Computing, Special Issue on Scalability**, April 1994.
6. Served as co-chair and vice-chair for many conferences and workshops on parallel computing.
7. Invited panelist in many conferences including SIAM conference on parallel computing, Foundations of Data Organization and Algorithms, and InfoJapan.

ANANTH GRAMA
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

Education

Ph.D. in Computer Science from University of Minnesota, May 1996 (expected).

Research Publications

FIVE PUBLICATIONS MOST CLOSELY RELATED TO PROPOSED PROJECT

1. "Isoefficiency Function: A Scalability Metric for Parallel Algorithms and Architectures", Ananth Grama, Anshul Gupta, and Vipin Kumar, *IEEE Parallel and Distributed Technology, Special Issue on Parallel and Distributed Systems: From Theory to Practice*, August 1993.
2. V. Kumar, Ananth Grama, and V.N. Rao, "Scalable Load Balancing Techniques for Parallel Computers" in *Journal of Parallel and Distributed Computing*, Volume 22, Number 1, pages 60-79, July 1994.
3. Ananth Grama, Vipin Kumar and Ahmed Sameh, "Parallel Iterative Solvers and Preconditioners Using Approximate Hierarchical Methods", proceedings of the Copper Mountain Conference on Iterative Methods, April 1996, Copper Mountain, CO.
4. Ananth Y. Grama, Vipin Kumar and Ahmed Sameh, Parallel Matrix-Vector Product Using Approximate Hierarchical Methods, Proceedings of Supercomputing'95, December 1995, San Diego.
5. Ananth Y. Grama, Vipin Kumar and Ahmed Sameh, Scalable Parallel Formulations of the Barnes-Hut Method for n-Body Simulations, Proceedings of Supercomputing'94, November 1994, Washington D.C.

Books

1. Vipin Kumar, Ananth Grama, Anshul Gupta and George Karypis, "Introduction to Parallel Computing: Algorithm Design and Analysis", Benjamin-Cummings, 1994.

Honors

1. Awarded the "Doctoral Dissertation Fellowship" by the Graduate School of the University of Minnesota.
2. Awarded the "Doctoral Dissertation Award" by the Department of Computer Science, University of Minnesota, in recognition of the research work published in the paper "Scalable Load Balancing Techniques for Parallel Computers", Journal of Parallel and Distributed Computing, 1994.
3. Awarded Vice-Chancellors Gold Medal for best overall grades among all engineering majors of the university (University of Roorkee, Roorkee, India, June 1989).

Invited Talks

"Efficient Parallel Formulations of Hierarchical Methods and their Applications", **PetaFLOPS Summer Study**, Bodega Bay, CA.

GEORGE KARYPIS
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

Education

Ph.D. in Computer Science from University of Minnesota, May 1996 (expected).

Research Publications

<u>Five publications most closely related to proposed project</u>

1. "Parallel Multilevel Graph Partitioning", George Karypis and Vipin Kumar. In 10th International Parallel Processing Symposium, 1996.
2. " Analysis of Multilevel Graph Partitioning", George Karypis and Vipin Kumar. Supercomputing 95, December 1995, San Diego.
3. "Multilevel Graph Partitioning and Sparse Matrix Ordering", George Karypis and Vipin Kumar. International Conference on Parallel Processing, 1995.
4. "A High Performance Sparse Cholesky Factorization Algorithm for Scalable Parallel Computers", George Karypis and Vipin Kumar. Eighth Symposium on the Frontiers of Massively Parallel Computation, 1995.
5. "Scalable Parallel Formulation of Interior Point Algorithms", George Karypis, Anshul Gupta, and Vipin Kumar. Supercomputing 1994.

Books

1. Vipin Kumar, Ananth Grama, Anshul Gupta and George Karypis, "Introduction to Parallel Computing: Algorithm Design and Analysis", Benjamin-Cummings, 1994.

Honors

1. First Prize Award at European Supercomputing Conference, Mannheim SuParCup 95.
2. Cray Research Fellow for 1995–96.
3. Graduate School Fellow University of Minnesota for 1992–93.

- -