Final Report for
Cooperative Agreement
NASA Ames Research Center
NASA NCC 2-5285
Algorithms and Libraries

8/1/98 to 11/1/98
$50,000

Principal Investigator
Jack Dongarra
University of Tennessee, Knoxville
Computer Science Department

Technical Officer for the Cooperative Agreement
Subhash Saini
Numerical Aerospace Simulation Systems Branch, T27A-1

Grant Officer
Venoncia Braxton

This exploratory study initiated our inquiry into algorithms and applications that would benefit by latency tolerant approach to algorithm building, including the construction of new algorithms where appropriate. In a multithreaded execution, when a processor reaches a point where remote memory access is necessary, the request is sent out on the network and a context--switch occurs to a new thread of computation. This effectively masks a long and unpredictable latency due to remote loads, thereby providing tolerance to remote access latency.

We began to develop standards to profile various algorithm and application parameters, such as the degree of parallelism, granularity, precision, instruction set mix, inter--processor communication, latency etc. These tools will continue to develop and evolve as the Information Power Grid environment matures. To provide a richer context for this research, the project also focused on issues of fault-tolerance and computation migration of numerical algorithms and software.

During the initial phase we tried to increase our understanding of the bottlenecks in single processor performance. Our work began by developing an approach for the automatic generation and optimization of numerical software for processors with deep memory hierarchies and pipelined functional units. The production of such software for machines ranging from desktop workstations to embedded processors can be a tedious and time-consuming process. The work that was done helps in automating much of this process. We have concentrated our efforts on the widely used linear algebra kernels called the Basic Linear Algebra Subroutines (BLAS). In particular, the work conducted focused on general matrix multiplication, DGEMM. However much of the technology and methodology developed here can be applied to the other Level 3 BLAS. Moreover

the general strategy can have an impact on basic linear algebra operations in general and may be extended to other important kernel operations.

In our approach,which we call *ATLAS* (Automatically Tuned Linear Algebra Software), we isolated the machine-specific features of the operation to several routines, all of which deal with performing an optimized on-chip (i.e., in level 1 cache) matrix multiply of the form $C \leftarrow C + A*B$. This section of code is automatically created by the ATLAS code generator, which uses timings to determine the correct blocking and loop unrolling factors to perform an optimized on-chip multiply. The user may directly supply the code generator with as much detail as desired (i.e., the user may explicitly indicate the level 1 cache size, the blocking factor(s) to try, etc.); if such details are not provided, the generator determines appropriate settings via timings. The rest of the code does not change across architectures, other than perhaps including preprocessor information discovered by the code generator. The ATLAS method also handles blocking for higher level caches (if any), and the necessary overhead required to build the complete matrix-matrix multiply from the on-chip multiply.

Figure 1 shows double precision matrix multiply performance across multiple architectures for matrix order 500, comparing ATLAS's DGEMM with that provided by the vendor.

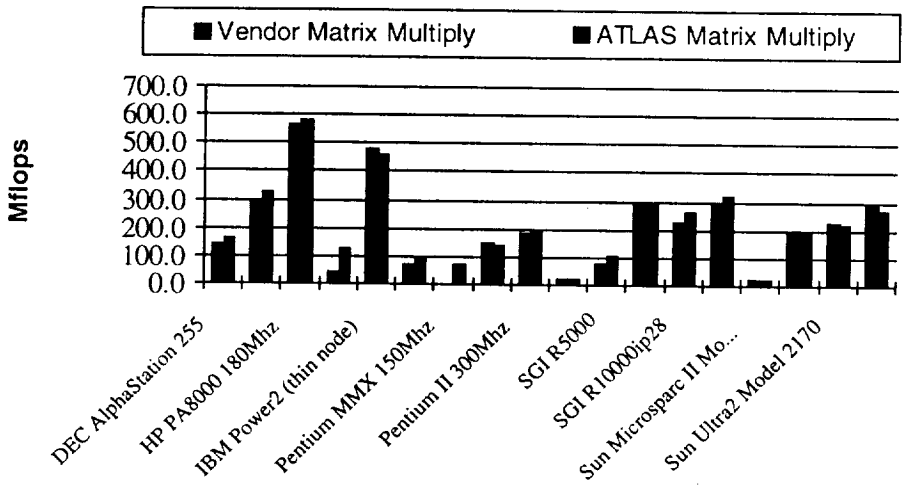

Figure 1

Based on the results we achieved in this study, we are planning to study other architectures of interest, including development of cost models, and developing code generators appropriate to these architectures. Future RISC processors with vector instructions might require loop lengths that match the optimal vector lengths, for instance. SMPs will require load balancing while avoiding "false sharing" of cache lines by different processors. Different ways of thread management will also have to be

considered. On clusters of SMPs load balancing where one process on each SMP is responsible for off-SMP communication and less floating point work will be considered.

Following up on the results investigation, we are working now on extending this work to the level 2 BLAS, as well as adding threading as part of the package. We have initial results for these extensions. We are also planning to develop and refine algorithms that exploit sparse BLAS. Sparse matrix-vector multiplication is a essential kernel in most iterative algorithms for large matrix problems. Optimizing its performance requires a number of both architecture and matrix dependent transformations. We will study how to extend ATLAS to optimize sparse-matrix vector multiplication, where the optimizations may depend on the sparsity structure.

The results of this initial phase of study have demonstrated the ability of our ATLAS method to produce a highly optimized matrix multiply for a wide range of architectures based on a code generator that probes and searches the system for an optimal set of parameters. This avoids the tedious task of generating by hand routines optimized for a specific architecture. We have shown that these ideas can be expanded to cover not only the Level 3 BLAS, but Level 2 BLAS as well. In addition we have shown that there is scope for additional operations beyond the BLAS, such as sparse matrix vector multiplication, and FFTs.