Foundations of Programming Linear Algebra Algorithms on SMP and Multicore Systems

Robert van de Geijn
Kazushige Goto
The University of Texas at Austin

This project pursues the theory and practice of implementing algorithms for dense linear algebra operations on SMP and multicore systems. It will develop algorithms, abstractions, APIs, implementations, software architectures, analyses, and simulators. The successful completion of the project will improve the understanding of the subject and will provide tools for implementing scalable libraries and pedagogical tools. Although library production is not a major focus of the project, the libraries that result will be made available to the scientific computing community. In addition, it is expected to provide important insights into the design and use of current and multicore processors.

The use of Symmetric MultiProcessor (SMP) systems has become the norm in scientific computing. Desktop workstations on which applications are initially developed now typically have up to four CPUs. SMP systems with 16 to 256 CPUs are now available from vendors like Dell, HP, NEC, SGI, and Sun, and are widely used at universities, at government labs, and in industry. Distributed memory architectures now have SMPs as individual nodes. More recently, architectures where individual processors have multiple cores were announced, so that SMP-like parallel computing will occur within an individual chip. Thus, even the average application programmer will need to have the understanding and the access to tools that allow him/her to develop shared memory parallel codes. The most widely used tools for scientific computing are the Basic Linear Algebra Subprograms (BLAS) and LAPACK libraries. Thus, a thorough understanding of the scalable implementation of such libraries on SMP systems and multicore processors is of fundamental importance, as is the development of the libraries themselves.

Optimal implementation of dense linear algebra operations on SMP systems and/or multicore processors is a complex problem. In order to attain near-optimal performance on shared memory architectures with multi-level memories, it is essential to be able to use loop-based algorithms. (So-called cache-oblivious algorithms are inherently suboptimal since striding through operands in blocks that obey the size of the caches is essential for performance.) For different levels of the hierarchy, different algorithms for the same operation must be employed. Copying is required to keep data in the various caches and/or to avoid TLB misses. Duplication of copying overhead by different threads requires implementations to break through boundaries between layers like LAPACK and the BLAS. And finally there is the scheduling of tasks to processors. We note that the recently funded Sca/LAPACK project chose not to address this problem, even though progress in this area is likely to have a broader impact on users of LAPACK than the extension of functionality that that project does pursue.

The team assembled at UT-Austin has unique qualifications for tackling this complex problem: The FLAME project, funded by NSF, has produced a methodology for deriving families of algorithms for dense linear algebra operations. A mechanical system for deriving all loop-based algorithms greatly reduces the development time. APIs for implementing the resulting algorithms allow code to closely resemble the (proven correct) algorithms, so that programming “bugs” are not easily introduced in the translation. CoPI Kazushige Goto is known for his high-performance implementations of BLAS libraries, which are widely used by the computational science community. Extensive experience gained from the development of PLAPACK for distributed memory architectures has applicability to the development of such libraries for SMP systems and/or multicore processors.

The intellectual merit of the project lies with the knowledge that will be gained regarding the parallel implementation of linear algebra libraries and applications in general on SMP systems and multicore processors. In addition, insights are expected to affect future design of architectures with large numbers of cores on a single chip. The broader impact lies with the tools, in form of libraries, that will be made available to the computational science community, and the pedagogical benefits.

1 Introduction

With the advent of multicore architectures, shared memory architectures will reach new levels of parallelism. Shared memory architectures with 256 or more CPUs are already available, and the possibility that each CPU will have multiple cores means that scalable parallelism on such architectures will be the most important technical problem to be overcome. While the hardware is already available, software lags far behind.

The most widely used tools for scientific computing are the Basic Linear Algebra Subprograms (BLAS) and LAPACK libraries. Thus, a thorough understanding of the scalable implementation of such libraries on SMP systems and multicore processors is of fundamental importance, as is the development of the libraries themselves.

Remark 1. *The proposed project is complementary to the recently NSF funded Sca/LAPACK project. The proposal for that project does not mention SMP parallelism nor multicore processors. Moreover, it rigidly adheres to the original software architecture of LAPACK, a design choice that makes many of the studies and solutions that we propose difficult within LAPACK.*

1.1 Relevant results from the FLAME project

Remark 2. *Throughout this proposal, we will use a simple concrete example to illustrate the issues. The operation we chose is the symmetric rank-k update (*SYRK*): $C := AA^T + C$, where C is a symmetric matrix of which only the lower triangular part is stored and updated. While the example is simple, the methodology applies to much more complex operations.*

Recent research of ours, sponsored by NSF, has led to the following insights:

- **Representing algorithms.** Loop-based dense linear algebra algorithms can be expressed in a clear way that captures how they are described in a classroom setting, without the use of obfuscated indexing. The notation is illustrated in Fig. 1 for algorithms for computing SYRK [19, 17, 5].
- **Multiple algorithms.** A systematic derivation process yields families of algorithmic variants for each linear algebra operation [19, 17, 5, 25]. In Fig. 1 four variants are given for SYRK.
- **An algorithm for every occasion.** Each algorithm has different performance characteristics on different architectures. More importantly, on different architectures and/or dependent on problem size, different algorithmic variants achieve better performance. This is illustrated later in Fig. 3.
- **Representing algorithms in code.** By using APIs that capture the format of the algorithms, "bugs" that are often introduced when translating algorithms to code can be avoided [6, 26, 37]. An API for the C programming language, FLAME/C, is illustrated in Fig. 2(left).

1.2 Programming Dense Linear Algebra Algorithms on SMP Systems

The above observations have led to insights that form the foundation for the current proposal. These preliminary insights have been documented in

- Tze Meng Low et al. "Parallelizing FLAME Code with OpenMP Task Queues." ACM Trans. Math. Soft., in revision [25].
- Tze Meng Low et al. "Extracting SMP Parallelism for Dense Linear Algebra Algorithms from High-Level Specifications." PPoPP05, to appear [27].

FLAME/C code inherently avoids the use of indices. As a result, standard constructs that are part of OpenMP [30, 21] cannot be used to elegantly express tasks and/or that iterations are independent. A new model, workqueuing, has been proposed for the next OpenMP standard, instantiated as the task queue construct [34, 36]. Adding this construct to FLAME/C code is illustrated in Fig. 2(right). Some comment about the code:

```
Algorithm: $C := \textsc{Syrk\_blk\_var1\_2}( A, C )$
Partition $C \rightarrow \left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right)$ , $A \rightarrow \left(\begin{array}{c} A_T \\ \hline A_B \end{array}\right)$
    where $C_{TL}$ is $0 \times 0$, $A_T$ has 0 rows

while $m(C_{TL}) < m(C)$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} C_{00} & C_{01} & C_{02} \\ \hline C_{10} & C_{11} & C_{12} \\ C_{20} & C_{21} & C_{22} \end{array}\right)$,
        $\left(\begin{array}{c} A_T \\ \hline A_B \end{array}\right) \rightarrow \left(\begin{array}{c} A_0 \\ \hline A_1 \\ A_2 \end{array}\right)$
    where $C_{11}$ is $b \times b$ , $A_1$ has $b$ rows
    ------------------------------------------------------------
    Variant 1:                               | Variant 2:
    $C_{10} := A_1 A_0^T + C_{10}$           | $C_{21} := A_2 A_1^T + C_{21}$
    $C_{11} := A_1 A_1^T + C_{11}$           | $C_{11} := A_1 A_1^T + C_{11}$
    ------------------------------------------------------------
    Continue with
        $\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} C_{00} & C_{01} & C_{02} \\ C_{10} & C_{11} & C_{12} \\ \hline C_{20} & C_{21} & C_{22} \end{array}\right)$,
        $\left(\begin{array}{c} A_T \\ \hline A_B \end{array}\right) \leftarrow \left(\begin{array}{c} A_0 \\ A_1 \\ \hline A_2 \end{array}\right)$
endwhile
```

```
Algorithm: $C := \textsc{Syrk\_blk\_var3\_4}( A, C )$
Partition $C \rightarrow \left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right)$ , $A \rightarrow \left(\begin{array}{c} A_T \\ \hline A_B \end{array}\right)$
    where $C_{BR}$ is $0 \times 0$, $A_B$ has 0 rows

while $m(C_{BR}) < m(C)$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} C_{00} & C_{01} & C_{02} \\ C_{10} & C_{11} & C_{12} \\ \hline C_{20} & C_{21} & C_{22} \end{array}\right)$,
        $\left(\begin{array}{c} A_T \\ \hline A_B \end{array}\right) \rightarrow \left(\begin{array}{c} A_0 \\ A_1 \\ \hline A_2 \end{array}\right)$
    where $C_{11}$ is $b \times b$ , $A_1$ has $b$ rows
    ------------------------------------------------------------
    Variant 3:                               | Variant 4:
    $C_{21} := A_2 A_1^T + C_{21}$           | $C_{10} := A_1 A_0^T + C_{10}$
    $C_{11} := A_1 A_1^T + C_{11}$           | $C_{11} := A_1 A_1^T + C_{11}$
    ------------------------------------------------------------
    Continue with
        $\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|cc} C_{00} & C_{01} & C_{02} \\ \hline C_{10} & C_{11} & C_{12} \\ C_{20} & C_{21} & C_{22} \end{array}\right)$,
        $\left(\begin{array}{c} A_T \\ \hline A_B \end{array}\right) \leftarrow \left(\begin{array}{c} A_0 \\ \hline A_1 \\ A_2 \end{array}\right)$
endwhile
```

Figure 1: Four blocked algorithms for computing $C := AA^T + C$.

```c
int Syrk_blk_var2( FLA_Obj C, FLA_Obj A, int nb_alg )
{
  FLA_Obj CTL,   CTR,      C00, C01, C02,         AT,              A0,
          CBL,   CBR,      C10, C11, C12,         AB,              A1,
                           C20, C21, C22,                          A2;
  int b;

  FLA_Part_2x2( C,    &CTL, &CTR,
                      &CBL, &CBR,     0, 0, FLA_TL );
  FLA_Part_2x1( A,    &AT,
                      &AB,            0, FLA_TOP );


  while ( FLA_Obj_length( CTL ) < FLA_Obj_length( C ) ){
    b = min( FLA_Obj_length( CBR ), nb_alg );

    FLA_Repart_2x2_to_3x3( CTL, /**/ CTR,   &C00, /**/ &C01, &C02,
                          /*************/  /********************/
                                            &C10, /**/ &C11, &C12,
                           CBL, /**/ CBR,   &C20, /**/ &C21, &C22,
                           b, b, FLA_BR );
    FLA_Repart_2x1_to_3x1( AT,               &A0,
                         /* ** */          /* ** */
                                             &A1,
                           AB,               &A2,   b, FLA_BOTTOM );
    /*-----------------------------------------------------------*/

    FLA_Gemm( FLA_NO_TRANSPOSE, FLA_TRANSPOSE,
      ONE, A2, A1, ONE, C21 );
    FLA_Syrk( FLA_LOWER_TRIANGULAR, FLA_NO_TRANSPOSE,
      ONE, A1, ONE, C11 );

    /*-----------------------------------------------------------*/
    FLA_Cont_with_3x3_to_2x2( &CTL, /**/ &CTR,  C00, C01, /**/ C02,
                                                C10, C11, /**/ C12,
                              /*************/ /*****************/
                              &CBL, /**/ &CBR,  C20, C21, /**/ C22,
                              FLA_TL );
    FLA_Cont_with_3x1_to_2x1( &AT,                 A0,
                                                   A1,
                            /* ** */             /* ** */
                              &AB,                 A2,     FLA_TOP );
  }

}
```

```c
int OpenFLA_Syrk_blk_var2( FLA_Obj C, FLA_Obj A, int nb_alg )
{
  FLA_Obj CTL,   CTR,      C00, C01, C02,         AT,              A0,
          CBL,   CBR,      C10, C11, C12,         AB,              A1,
                           C20, C21, C22,                          A2;
  int b;

  FLA_Part_2x2( C,    &CTL, &CTR,
                      &CBL, &CBR,     0, 0, FLA_TL );
  FLA_Part_2x1( A,    &AT,
                      &AB,            0, FLA_TOP );

#pragma intel omp parallel taskq {
  while ( FLA_Obj_length( CTL ) < FLA_Obj_length( C ) ){
    b = min( FLA_Obj_length( CBR ), nb_alg );

    FLA_Repart_2x2_to_3x3( CTL, /**/ CTR,   &C00, /**/ &C01, &C02,
                          /*************/  /********************/
                                            &C10, /**/ &C11, &C12,
                           CBL, /**/ CBR,   &C20, /**/ &C21, &C22,
                           b, b, FLA_BR );
    FLA_Repart_2x1_to_3x1( AT,               &A0,
                         /* ** */          /* ** */
                                             &A1,
                           AB,               &A2,   b, FLA_BOTTOM );
    /*-----------------------------------------------------------*/
#pragma intel omp task captureprivate( A2, A1, C21, C11 ){
    FLA_Gemm( FLA_NO_TRANSPOSE, FLA_TRANSPOSE,
      ONE, A2, A1, ONE, C21 );
    FLA_Syrk( FLA_LOWER_TRIANGULAR, FLA_NO_TRANSPOSE,
      ONE, A1, ONE, C11 );
} /* end task */
    /*-----------------------------------------------------------*/
    FLA_Cont_with_3x3_to_2x2( &CTL, /**/ &CTR,  C00, C01, /**/ C02,
                                                C10, C11, /**/ C12,
                              /*************/ /*****************/
                              &CBL, /**/ &CBR,  C20, C21, /**/ C22,
                              FLA_TL );
    FLA_Cont_with_3x1_to_2x1( &AT,                 A0,
                                                   A1,
                            /* ** */             /* ** */
                              &AB,                 A2,     FLA_TOP );
  }
}   /* end of taskq */
}
```

Figure 2: Implementations of blocked Variant 2. *Left:* FLAME/C. *Right:* OpenFLAME (FLAME/C annotated with `taskq` pragmas).

- The `task` block is created by `#pragma intel omp parallel task`. This creates a queue to which tasks can be added. As threads become idle they will obtain tasks from this queue and execute them.
- Tasks are submitted to the queue by `#pragma intel omp task captureprivate( A0, A1, C10, C11 )`.

We would like to think that the approach is powerful, yet elegant and simple.

In Fig. 3 we show performance attained on a 4 CPU Itanium2 (1.5 GHz) server. The top of each graph represents peak performance on the system. Performance for various numbers of threads are reported. Very good performance and scaling is achieved, especially for larger matrices.

The code in Fig. 2(right) can be modified in a number of ways: The single task in the body of the loop (marked as "one loop/one task" in the graphs) can be split into two separate tasks (one for `FLA_Gemm` and one for `FLA_Syrk`). Also, one can split the loop into two loops, where the first schedules all the calls to `FLA_Gemm` and the second schedules calls to `FLA_Syrk`, all on the same task queue (marked as "two loops" in the graphs).

The graphs for Variant 2 and 3 respond to the corresponding algorithms in Fig. 1. (They perform better than Variants 1 and 4, for reasons that are explained in [25].) Variant 3 schedules the tasks in reverse order from Variant 2. The performance is much more erratic for Variant 3 since it schedules large `FLA_Gemm` related tasks last, which creates severe load imbalance as threads complete their final tasks. The "two loops" variation performs better since the smaller `FLA_Syrk` tasks (which all represent equal work) are scheduled last, smoothing imbalances encountered in the `FLA_Gemm` tasks. This illustrates the importance of considering multiple algorithmic variants and/or the importance of careful scheduling of tasks.

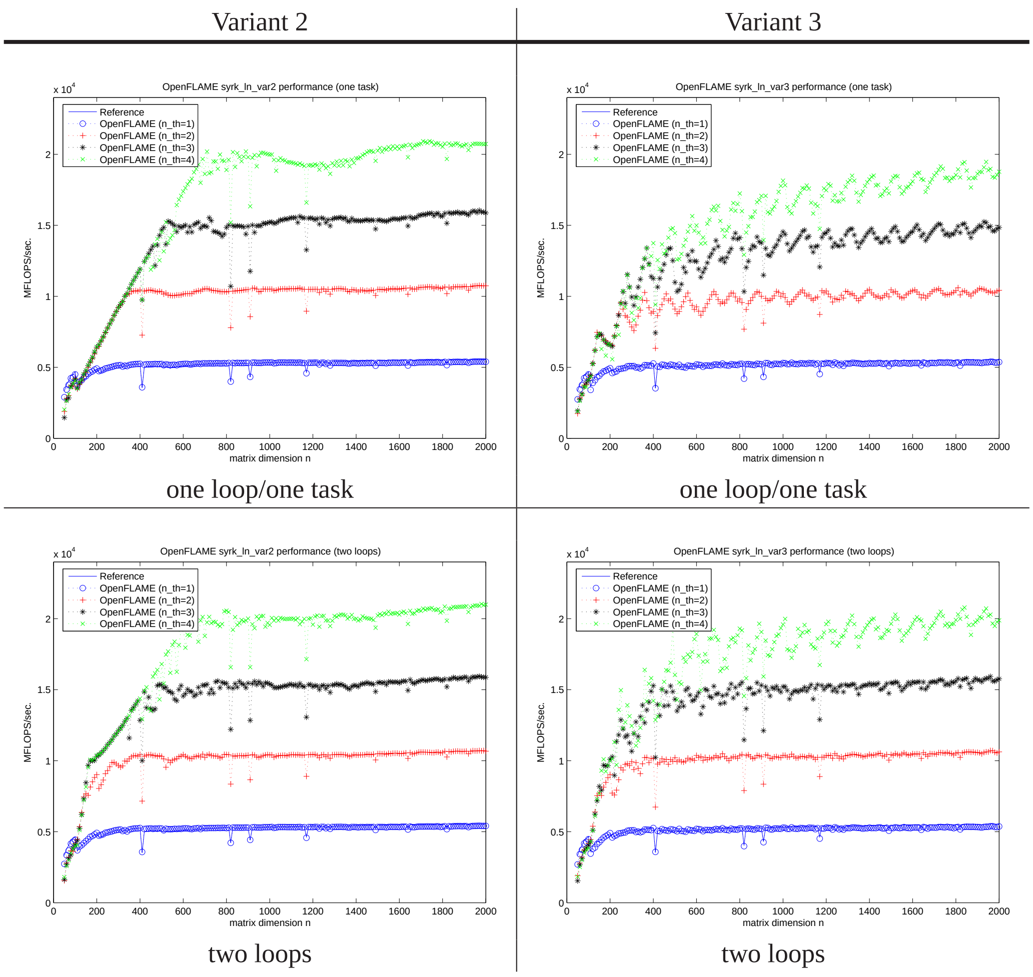

Figure 3: Performance of Variants 2 and 3.

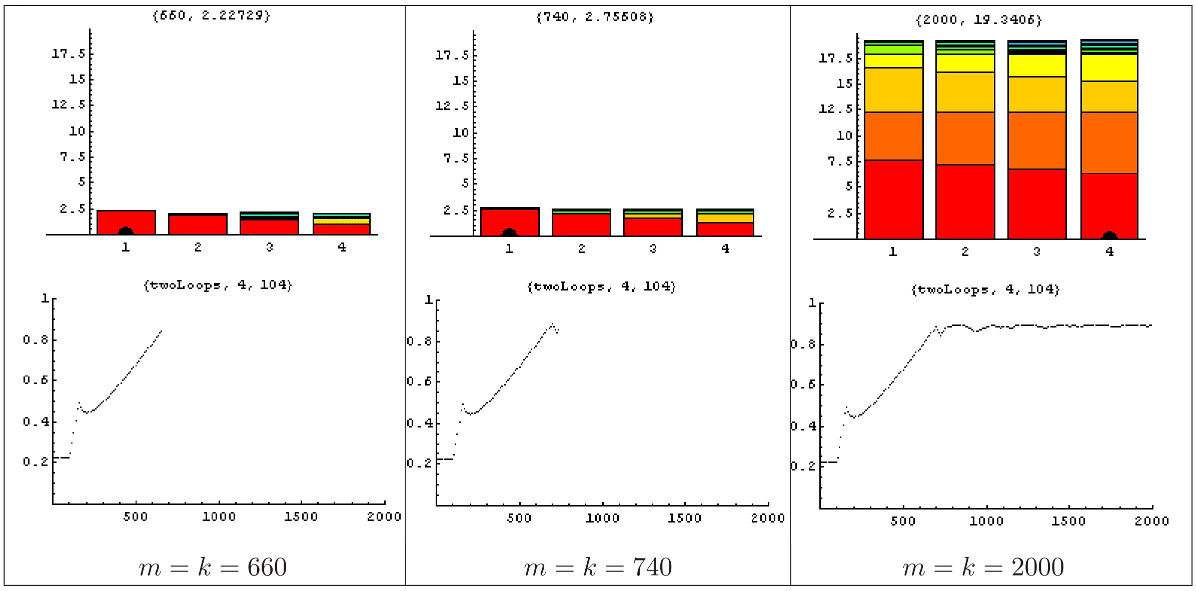

Figure 4: Simulation results for an SMP with 4 CPUs. For this simulation, the rate at which each task was performed was set at the rate attained per thread by the algorithm as reported in Fig. 3. *Top:* scheduling of tasks to threads. Threads are listed along the x-axis. Each box represents a task. The height of the box is proportional to amount of work of task. *Bottom:* Attained fraction of peak predicted by scheduling. *Left:* First task dominates computation. *Middle:* First thread gets more than one task. *Right:* Large number of smaller tasks, queued later, help load balance. Notice that the three sets of graphs from left to right represent snapshots in an animation that shows how tasks are scheduled to threads as larger and larger problem sizes are executed (top), and what the resulting predicted fraction of peak is (bottom).

Remark 3. *The example uses the* `taskq` *pragma which is only supported by the Intel compiler at this time (to our knowledge). Notice that it is the workqueuing model that is important here to our research. The task queue implementation of this model allowed us to prototype our ideas and show preliminary results. Later in the proposal it will be made clear that as part of the research we intend to implement the workqueuing model via POSIX threads [32] specifically for our needs, since it will allow us to perform experiments that cannot be conducted easily via the* `taskq` *construct, and it will make our solution portable to all SMP platforms.*
We have been told (see supporting letter from Dr. Mattson) that task queues are likely to be included in OpenMP 3.0. However, details of the semantics of the construct have not been finalized. Much work is expected on this issue over the next year. Thus, timely funding of the proposed work related to the workqueuing model will likely impact OpenMP 3.0.

1.3 Objectives of the proposed research

The objective of the proposed project is to

- Thoroughly develop theory and practice for the problem of building scalable dense linear algebra libraries for SMP and multicore systems.
- Build a complete library to demonstrate the developed techniques as well as to develop a theory and simulators for predicting the performance of the routines that are part of the resulting library.
- Impact the design of SMP systems and multicore processor by providing a rich set of algorithms, implementations, analytical results, and simulators. Through collaboration with the TRIPS project [8] at UT

and with industry we will have access to simulators for future multicore architectures. This will allow our work to be relevant to future architectures while simultaneously providing valuable feedback on the design of future multicore systems.

The contribution to computer science that will result from the successful completion will be in the new understanding that will result, not only of the issues surrounding the parallel implementation of this class of operations, but also of other operations. The broader impact to the computational science community will be the libraries that will be made available to the community.

1.4 The team

The team at UT-Austin is uniquely qualified to pursue the proposed project. It has strength in library development at all levels, including

- The low-level linear algebra kernels that carefully schedule movement between cache layers and registers [15, 14]. Kazushige Goto is the world's leading expert on the high-performance implementation of such kernels on essentially all current architectures.
- The theory and practice of high-performance matrix-matrix multiplication. Theoretical results to explain current practice, including those used by Kazushige Goto, were published by Gunnels, Henry, and van de Geijn [18].
- The implementation of BLAS libraries in terms of matrix-matrix multiplication [22]. The FLAME project has identified and implemented all practical sequential algorithms for layering BLAS operations on top of matrix-matrix multiplication. Prior to this, distributed memory implementations were also pursued as part of the PLAPACK project [37, 16, 10].
- The implementation of LAPACK-level operations in terms of the BLAS. The FLAME project has identified and implemented all practical sequential algorithms for a cross-section of operations supported by LAPACK.
- The parallel implementation of LAPACK-level operations on SMP and distributed memory architectures. The cited preliminary research provides initial evidence that the FLAME approach can be extended to SMP and multicore systems in an elegant and effective fashion. Our experience with the PLAPACK project will allow us to take that one step further to distributed memory architectures with SMP and/or multicore nodes. However, distributed memory parallelism is not a major focus of this research. What is important is that we will show that insights into scalable implementation on distributed memory architectures also apply to scalability on SMP systems.

In all of these cases, implementations have been demonstrated to rival or surpass the best known implementations. It is the vertical integration of expertise at all these levels that facilitates the proposed project.

1.5 Access to interesting architectures

A large cross-section of SMP systems will be available for the proposed research. Hewlett-Packard has donated three multi-CPU Itanium2 servers to the FLAME project (including one 4-CPU server). Relevant to the proposed project, the Texas Advanced Computing Center (TACC) of UT-Austin currently has a 224 processor IBM Power4 system (with individual nodes with up to 32 processors). Industrial partners will provide access to larger SMP systems (NEC already provides us access to 16-CPU Itanium2 servers).

Finally, the team will have access to advanced prototype systems: As part of the TRIPS project at UT-Austin, a prototype shared address space, distributed memory multiprocessor system with dual-core processors will become available in Spring, 2006 [8]. The system will consist of up to 32 dual-processors, each with 16 FPUs and up to 4 simultaneously executing threads for a theoretical maximum of 256-way parallelism and a peak throughput of up to 545 GFlops.

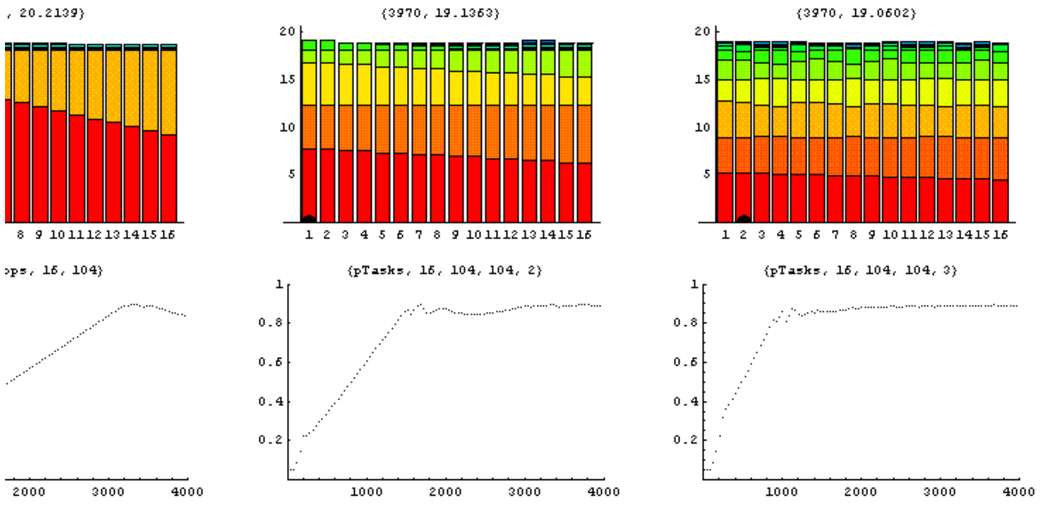

Figure 5: Simulation results for an SMP with 16 CPUs. Again, the rate at which each task was performed was set at the rate attained per thread by the algorithm as reported in Fig. 3. *Left:* "two-loop" variation from before. *Middle:* each call to `FLA_Gemm` is split into two tasks. *Right:* each call to `FLA_Gemm` is split into three tasks.

1.6 Industry participation

The project has established collaborations with vendor library groups, including those at Intel, NEC, HP, IBM, and National Instruments (NI). NI provides technical and financial support (see letter). NEC provides technical and financial support as well as access to hardware (see letter). HP has provided technical support and hardware to the project. IBM provides technical support (see letter) Technical expertise regarding the programming of future architectures with many cores per processor will be provided by Dr. Tim Mattson at Intel (see letter).

2 Research and Development Questions

The discussion so far sets the stage for a discussion of research and development questions.

2.1 Does the approach support a broad class of dense linear algebra operations on SMP systems?

Parallelization for SMP systems with large numbers of CPUs shares a lot with parallelization for distributed memory systems. The primary difference is that rather than communicating data, data moves from memory into the caches of individual CPUs.

From many projects related to the parallelization of dense linear algebra algorithms for distributed memory architectures [20, 9, 24, 37, 12], we have learned that they require two types of communications:

- Duplication of data, after which processors can proceed with independent computation.
- Reduction of data, upon completion of independent computation on separate processors.

The way the `taskq` pragma is used in our example captures the first: Duplication of data happens as it is being used, and the task queue supports the independent computation. In [25] we show how reduction of contributions from different threads can be accommodated. Given that these two constructs support most of the functionality of LAPACK in the ScaLAPACK [9] and PLAPACK [37] packages for distributed memory architectures, we are confident that the proposed approach is broadly applicable.

```
#pragma intel omp parallel taskq
 {
 while ( FLA_Obj_length( AT ) < FLA_Obj_length( A ) ){
   b = min( FLA_Obj_length( AB ), nb_alg );

   FLA_Repart_2x1_to_3x1( AT,                &A0,
                       /* ** */            /* ** */
                                             &A1,
                          AB,                &A2,        b, FLA_BOTTOM );
   FLA_Repart_2x2_to_3x3( CTL, /**/ CTR,       &C00, /**/ &C01, &C02,
                       /* ************* */   /* ******************* */
                                               &C10, /**/ &C11, &C12,
                          CBL, /**/ CBR,       &C20, /**/ &C21, &C22,
                          b, b, FLA_BR );
   /*------------------------------------------------------------*/

   b2 = FLA_Obj_length( A2 )/2;
   FLA_Part_2x1( A2,     &A2_T,
                         &A2_B,            b2, FLA_TOP );
   FLA_Part_2x1( C21,    &C21_T,
                         &C21_B,           b2, FLA_TOP );
     #pragma intel omp task captureprivate(A2_T, A1, C21_T)
     {
     /*    C1 = alpha * A1 * B' + C1;    */
     FLA_Gemm( FLA_NO_TRANSPOSE, FLA_TRANSPOSE,
               ONE, A2_T, A1, ONE, C21_T );
     }
     #pragma intel omp task captureprivate(A2_B, A1, C21_B)
     {
     /*    C1 = alpha * A1 * B' + C1;    */
     FLA_Gemm( FLA_NO_TRANSPOSE, FLA_TRANSPOSE,
               ONE, A2_B, A1, ONE, C21_B );
     }
   /*------------------------------------------------------------*/
   FLA_Cont_with_3x1_to_2x1( &AT,                A0,
                                                 A1,
                           /* ** */           /* ** */
                             &AB,                A2,      FLA_TOP );
   FLA_Cont_with_3x3_to_2x2( &CTL, /**/ &CTR,       C00, C01, /**/ C02,
                                                    C10, C11, /**/ C12,
                           /* ************** */  /* ****************** */
                             &CBL, /**/ &CBR,       C20, C21, /**/ C22,
                             FLA_TL );
 }
```

Figure 6: Loop that schedules FLA_Gemm tasks for Variant 2, with the FLA_Gemm call split into two smaller tasks. A generalization would split each FLA_Gemm into an arbitrary number of tasks via a loop.

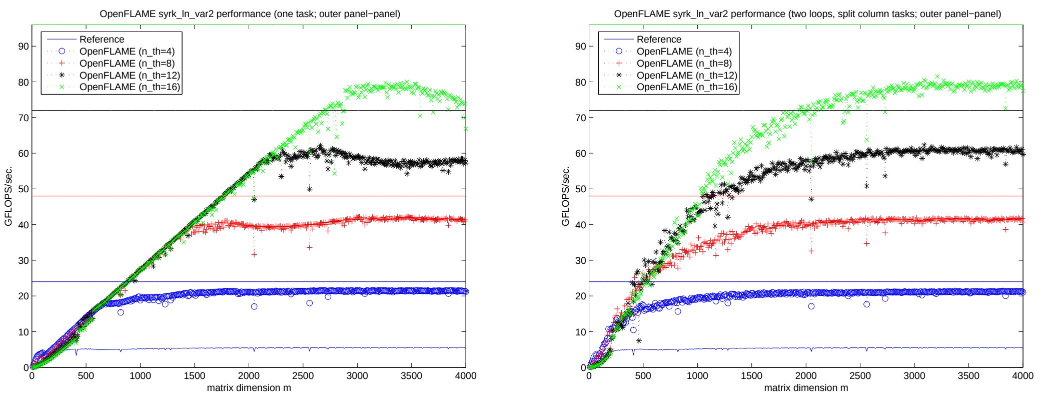

Figure 7: Performance of SMP implementations of SYRK (Variant 2). Here n_th indicates the number of threads used in the experiment. *Left:* 1D work decomposition. *Right:* 2D work decomposition.

2.2 Does the methodology scale?

In Fig. 7 (left) the performance on a 16 CPU Itanium2 server is given. Again, good performance is reported for large problems, but we now notice that the problem size is very large before the asymptote is reached. The simulation in Fig. 5 (left) shows that problem sizes must be very large before the first task is no longer dominant. (One can theoretically show that the matrix size for which the performance curve attains the asymptote is directly proportional to the number of CPUs being used, for this approach to parallelization.)

The simple fix is to partition each costly matrix-matrix multiplication task into r smaller tasks. Code for this when $r = 2$ is illustrated in Fig. 6. The effect on performance when $r = 2$ is shown in Fig. 7 and simulated for $r = 2, 3$ in Fig. 5. The simulation shows the ramp-up is roughly r times faster, for $r = 2, 3$.

More extensive experiments with the simulator have shown that r is optimal, under idealized circumstances, when $r \approx \sqrt{p}$ where p equals the available number of processors. For those of us who have experience with distributed memory architectures this is not surprising: On those architectures, processors must be arranged into a 2D mesh in order to attain scalability. Theoretical results, similar to those developed for distributed memory algorithms [20, 35], need to be established.

2.3 Can theoretical models and/or simulations explain the observed performance?

There is clear discrepancy between the simulated results in Fig. 5 and the observed results in Fig. 7. In the implementation and simulation there are at least three sources of inefficiency/error:

- The implementation must be refined. There are certain data copies and transpositions that occur inside highly optimized matrix-matrix multiplication kernels, to which `FLA_Gemm` is a wrapper. By splitting a single `FLA_Gemm` call into multiple calls, these data movements, which represent pure overhead, are redundantly executed. In addition, these data movements are extremely memory bandwidth intensive, and the fact that they are executed simultaneously likely degrades the performance of each individual data movement. This can be resolved by calling lower level kernels that were used to implement the BLAS routine `dgemm`, to which `FLA_Gemm` is linked.
- The simulator must be refined. There are many ways in which this can be done: The cost of an individual task can be more accurately predicted by storing a table of performance data for individual calls to `FLA_Gemm` and `FLA_Syrk`, for different problem sizes. In addition, we believe that the GOTOBLAS implementation of these kernels, implemented by CoPI Kazushige Goto, can be very accurately modeled due to the highly modular nature of the implementation [14, 15].
- Memory access conflicts must be taken into account.

2.4 At what layer of the library should SMP parallelism be exposed?

The lack of discussion of SMP parallelism in the recently published "LAPACK 2005 Prospectus" [11] leads us to believe that that project will extract SMP parallelism at the BLAS level.

Remark 4. *Street wisdom says that better performance and/or scalability is attained if SMP parallelism is extracted at higher levels than the BLAS.*

The previous example, although itself a BLAS operation, appears to support this to some degree. The most elegant way of coding it is NOT to explicitly split the call to `FLA_Gemm` into multiple calls as in Fig. 6, or to include it as a loop so that it can be split into many calls. Rather, the code should remain as in Fig. 2(right) and the desired parallelism should occur inside the call to `FLA_Gemm`. However, this means that the number of tasks into which the matrix-matrix multiply should be partitioned needs to be communicated to `FLA_Gemm`, which is a general purpose routine. In this case, we have argued that approximately $\sqrt{p}$ tasks should be created, where p is the number of available CPUs. But under different circumstances we have encountered the number of tasks created by `FLA_Gemm` should be p, or it itself should be implemented using a 2D partitioning of work (e.g., when `FLA_Gemm` is called by a right-looking LU factorization to perform a rank-k update).

What this suggests is that a higher level routine (a level 3 BLAS routine built upon matrix-matrix multiplication, or at a higher level yet, an LAPACK routine built upon BLAS) must be able to indicate to lower level routines what partitioning of work to use. This may suffice, or it may be necessary to express parallelism at that higher level explicitly.

Remark 5. *An extensive study is in order to develop theory, practice, and library constructs.*

Remark 6. *The point is that our framework will allow us to study this issue in detail, since all algorithms can be derived using the FLAME approach, implemented using FLAME/C, OpenFLAME, and PLAPACK, and performance can then be measured for the different implementations. Hopefully, our simulators will be able to give us additional insight.*

2.5 What if task queues don't make it into the OpenMP standard?

Although we initially advocated that all our development should be through task queues as implemented by the Intel compiler, we intend to pursue our own implementation of the workqueuing model, specific to our needs. There are a number of reasons for this:

- Currently the OpenMP task queue construct is not part of the standard and is only supported (to our knowledge) by the Intel C and Fortran compilers.
- Experimentation on other platforms is required so as not to skew the conclusions of our study in favor of Intel processor-based platforms.
- There are a number of studies that we cannot conveniently pursue through the task queue implementation provided by Intel. For example, experimentation with our simulator shows that a simple ordering of tasks in the task queue from largest task to smallest task provides good load balance for our example. We note that the theory of bean-packing indicates this to be an optimal *heuristic*. The question is whether this should be accomplished by, as in the case of the SYRK example, reversing the direction of the loop or splitting the loop into two loops. An alternative is to order the tasks in the task queue. A mechanism that would accomplish this with minimal developer intervention would require us to implement the workqueuing model ourselves. A first-order approximation of the cost of a task could be the floating point operation count.

Remark 7. *Insights that are gained in answering these questions will provide feedback to the OpenMP Forum regarding the* `taskq` *construct. It is our hope to influence the semantics of that construct, and possibly addition constructs yet to be identified, so that our best implementations can be supported solely by adding OpenMP directives.*

2.6 How does one optimize given a (potentially) large parameter space?

As part of the FLAME project, the mechanical derivation of algorithms and their cost is being pursued. If successful, it will be closed form estimates of the cost that we intend to use to determine optimal strategies for SMP systems. Another alternative is to use empirical optimization systems [38, 7]

2.7 What are the implications for multicore processors?

Multicore systems are already available from Intel, AMD, Sun, and IBM. It is speculated that individual processors with up at 128 cores may become available within ten years. Such systems may need to be programmed using a mix of techniques borrowed from shared and distributed memory parallel computing. For example, data could be shared between the caches of cores via collective communication. We note that we have published extensively on the subject of collective communication on distributed memory architectures [28, 23, 3, 1, 2, 4].

The experience we will gain from experiments with SMP systems, including those with multicore processors, in combination with the simulators to be developed as part of the project, will allow us to impact the design and use of multicore systems with large number of cores.

3 Approach

One definition of *science* is *knowledge that has been reduced to a system*. The natural steps toward making the parallel implementation of dense linear algebra algorithms on SMP and multicore systems a science is to first perform initial experiments, to gain insights into what is observed through simulators, to explain what is observed (make knowledge systematic) through models and theories, and to then verify these insights through a subsequent, grander experiment. Below we outline how we intend to pursue all of these steps.

3.1 Prototype implementation of a cross-section of operations

Objective To verify that the workqueuing model allows implementation of a large class of dense linear algebra operations. To study advantages and disadvantages of exposing SMP parallelism at different levels of the library. To study the necessity of breaking through library layers in order to avoid duplicating overhead within work assigned to different threads.

Status Prototype implementations of SYRK and GEMM have been used to study the workqueuing model using the proposed OpenMP task queue construct. Exposure of parallelism at different levels of these operations (e.g., partially within the GEMM operation called from the SYRK implementation) has been studies. Highly optimized kernels (including data copying and low-level matrix-matrix multiplication kernels) by Kazushige Goto have been componentized on a few architectures so that they can be called from FLAME/C code. This sets the stage for studies of how duplicate work can be avoided by breaking through library layers.

Proposed work

Workqueuing: An infrastructure for expressing workqueuing will be developed independent of the task queuing mechanism supported by Intel's OpenMP compiler.

Low-level kernels: Optimized kernels will be developed for multicore processors. Componentized kernels will be developed for all platforms. A standardized interface to these low-level computational and data copying kernels will be proposed. Performance impact of this componentized approach vs. the best current implementations will be quantified.

SMP GEMM: Optimal implementation by breaking through library layers will be studied. Optimal algorithms and 2D partitioning of work will be studied, for various shapes of operands. (E.g., rank-k update requires a different parallelization than a matrix-panel (of columns) multiplication []). An interface for indicating how much parallelism to extract through GEMM will be proposed.

SMP SYRK and TRSM: The study of SMP implementation of SYRK will be completed and a similar study will be conducted for the (important) triangular solve with multiple right-hand sides (TRSM) operation. This will require the completion of the inner kernels and GEMM related work. It will give insight into how parallelism can be split between one layer (the SYRK and TRSM algorithms) and a lower level operation (GEMM). All algorithms for SYRK and TRSM will be developed and parallelized for SMP systems, with variants that expose parallelism at different levels and that break through library layers in different ways.

SMP CHOL, LU, and LUPIV: The study of SMP implementation of Cholesky factorization, LU factorization, and LU factorization with partial pivoting will be conducted. This will build on the above mentioned work. It will give further insight into how parallelism can be split between one layer (the factorization algorithm) and a lower level operation (the BLAS). All algorithms for these operations will be developed and parallelized for SMP systems, with variants that expose parallelism at different levels and that break through library layers in different ways.

3.2 Experimentation

Objective To gain empirical insight into the interaction between SMP parallelism exposed at different levels of the library, the breaking through memory layers, the expressiveness of the instantiation of the workqueuing model, and the combined effect on performance.

Status Limited experimentation has been performed.

Proposed Work Extensive experimentation with the implementations mentioned under **Prototype implementation of a cross-section of operations**.

3.3 Analysis

Objective To gain theoretical insight into the interaction between SMP parallelism exposed at different levels of the library, the breaking through memory layers, the expressiveness of the instantiation of the workqueuing model, and the combined effect on performance.

Status Simple models, which underlie the rudimentary simulator described below, have been developed.

Proposed work

Low level kernels: A model for predicting the cost of the low level kernels will be developed. We believe that the simplicity of the approach to the highest performing implementations, developed by Kazushige Goto, will allow us to accurately model these kernels.

BLAS and LAPACK level operations: We believe that the model of the low level kernels, and an understanding of how the different algorithmic variants utilize the low level kernels, will allow us to accurately model the GEMM, SYRK, CHOL, LU, and LUPIV operations.

Workqueuing A model of workqueuing will be developed that can incorporate the modeling of the individual operations mentioned above. Variations that expose SMP parallelism at different levels will be modeled so that advantages and disadvantages can be exposed.

3.4 Simulation

Objective To gain insight through simulation into the interaction between SMP parallelism exposed at different levels of the library, the breaking through memory layers, the expressiveness of the instantiation of the workqueuing model, and the combined effect on performance. To verify analytical insights.

Status We have developed a rudimentary simulator coded in Mathematica. Snapshots for different sized problems are given in Figs. 4 and 5. The simulator assigns tasks to threads and calculates the expected runtime for each task. Already much of the qualitative behavior in Fig. 3 can be explained with this simple simulator.

Proposed work The models developed under **Analysis** will be incorporated into the simulator so that

- The predicted performance by the model can be compared to observed performance.
- Insights can be gained regarding strategies for optimally combining algorithms and optimally exposing SMP parallelism at different levels of the library.

3.5 Design of software architecture

Objective To validate the insights gained from the initial experiments through application to a large portion of the BLAS and LAPACK, for SMP and/or multicore systems.

Status No progress yet.

Proposed work Implementation of all of the BLAS and most of LAPACK using the developed techniques. Validation of the modeling techniques through modeling and simulation of implementations of a broader set of operations. Assessment of performance and scalability benefits for these software libraries.

4 Team/Project Management

The project builds on the extensive experience of researchers at UT-Austin, both in the Department of Computer Sciences and at the Texas Advanced Computing Center (TACC).

- **Dr. Robert A. van de Geijn,** PI and Professor of Computer Sciences, will head the team.
- **Mr. Kazushige Goto,** Research Associate at TACC and CoPI, is known for developing the fastest BLAS libraries on a wide variety of architectures. He will assist in the design of the performance model, will implement low-level kernels, and will help design algorithms that break through memory layers in different ways.
- **Dr. Enrique Quintana-Ortí,** Associate Professor of Computer Science, University of Jaume I, Spain. Dr. Quintana has published extensively in the areas of high-performance computing, parallel computing, and algorithms for control theory. He is a frequent visitor to UT-Austin and coauthor on a large number of FLAME and PLAPACK related publications. As part of the project, Dr. Quintana will visit UT-Austin for one-month periods every year, funded by the grant.
- **One full-time programmer** will assist in the considerable coding effort.
- **One Graduate Research Assistant**. The more theoretical components of the proposed work will lead to a dissertation for a Ph.D. student to be supported by the proposed project.

5 Impact

The projects related to FLAME are having a broad impact on research in linear algebra libraries and computational science:

- **Impact on Linear Algebra Libraries:** The approach demonstrates and enables a deeper level of understanding of the algorithmic, analysis, and software engineering issues related to linear algebra libraries.
- **Impact on Applications:** The resulting libraries and tools benefit the high-performance computing community.
- **Impact on OpenMP:** As already mentioned, the proposed work related to workqueues will likely impact the semantics of OpenMP constructs that have been proposed for OpenMP 3.0.
- **Impact on Processor Architecture:** Our studies will likely provide valuable feedback on the design of multicore processors.
- **Impact on Education:** The simple interfaces and abstractions enable advanced topics in the field to be made accessible to new graduate students and even undergraduate students.

6 Dissemination of Results

The results of the proposed project will be disseminated through the usual means: refereed journal and conference papers, a working note series, an attractive project web site and code available on the project web site. In addition, the research team will continue to meet frequently with collaborators from industry and will organize minisymposia on the developed techniques and tools. Broader dissemination of information about the project's existence and progress will be accomplished through features written by TACC's science writer and printed and/or published on the web. The work is expected to yield at least one Ph.D. dissertation.

7 Related Work

Related work has been discussed throughout the proposal.

8 Proposer's Relevant Research

Proposer's relevant work has been discussed throughout the proposal.

Comparison with Long-term Goals of PI: It is the long-term goal of the PI to facilitate the evolution of programming practices in the area of numerical libraries from the relative ad hoc approaches used currently to ones that are based on sound scientific principles. The current proposal pursues that goal.

9 Results from Prior NSF Support

Since July 1, 2002, the FLAME project at UT-Austin has been funded by three NSF grants:

Robert van de Geijn: Award ACI- 0342369 for " Automatic Tools for Deriving, Analyzing, and Implementing Linear Algebra Libraries." Performance period: March 1 2004-Feb. 28 2007. Amount: $299,999.

Robert van de Geijn (with Anthony Skjellum): Award ACI-0305163 for "ALGORITHMS: Collaborative Research: A Systematic Approach to the Derivation, Representation, Analysis, and Correctness of Dense and Banded Linear Algebra Algorithms for HPC Architectures." Performance period: Aug. 2003-July 2006; Amount: $238,259.

Robert van de Geijn (with Anthony Skjellum): Award ACI-0203685 for "ALGORITHMS: Collaborative Research: New Contributions to the Theory and Practice of Programming Linear Algebra Libraries." Performance period: 7/1/02-6/31/03; Amount: $35,000.

The above three projects are closely related to each other and the proposed project. Two Ph.D. students, Paolo Bientinesi and Tze Meng Low, were supported in part by these projects and have advanced to candidacy in 2004. A third Ph.D. student, Zaiqing Xu, is supported by the project and is expected to reach candidacy by Summer 2006. A Masters students, Field Van Zee, is expected to complete a Masters degree by Spring 2006.

These projects have yielded a wealth of new contributions. The following publications acknowledge support from the grants:

Books

1. Robert van de Geijn and Enrique Quintana-Orti. *The Science of Programming Matrix Computations.* A contract has been offered by CRC Press.

Journal publications

1. Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique Quintana-Orti, and Robert van de Geijn. "The Science of Deriving Dense Linear Algebra Algorithms," *ACM Trans. Math. Soft.*, 31(1):1-26, March 2005.
2. Paolo Bientinesi, Enrique Quintana-Orti, and Robert van de Geijn. "Representing Linear Algebra Algorithms in Code: The FLAME APIs." *ACM Trans. Math. Soft.*, 31(1):1-26, March 2005.
3. Brian Gunter and Robert van de Geijn. "Parallel Out-of-Core Computation and Updating of the QR Factorization." *ACM Trans. Math. Soft.*, 31(1):1-26, March 2005.
4. Paolo Bientinesi, Inderjit S. Dhillon, and Robert A. van de Geijn. "A Parallel Eigensolver for Dense Symmetric Matrices Based on Multiple Relatively Robust Representations." SIAM Journal on Scientific Computing, to appear.
5. Gregorio Quintana-Orti and Robert van de Geijn. "Improving the Performance of Reduction to Hessenberg Form." *ACM Trans. Math. Soft.*, accepted.
6. Tze Meng Low, Kent Milfeld, Robert van de Geijn, and Field Van Zee. "Parallelizing FLAME Code with OpenMP Task Queues." *ACM Trans. Math. Soft.*, in revision.
7. Thierry Joffrain, Tze Meng Low, Enrique Quintana-Orti, Robert van de Geijn, and Field Van Zee. "On Accumulating Householder Transformations." *ACM Trans. Math. Soft.*, in revision.

Conference Publications

1. Tze Meng Low, Robert van de Geijn, and Field Van Zee. "Extracting SMP Parallelism for Dense Linear Algebra Algorithms from High-Level Specifications." PPoPP'05, to appear.
2. Paolo Bientinesi, John Gunnels, Fred Gustavson, Greg Henry, Margaret Myers, Enrique Quintana-Orti, and Robert A. van de Geijn. "Rapid Development of High-Performance Linear Algebra Libraries." Proceedings of PARA04, to appear.
3. Paolo Bientinesi, Sergey Kolos, and Robert A. van de Geijn. "Automatic Derivation of Linear Algebra Algorithms with Application to Control Theory." Proceedings of PARA04, to appear.
4. Paolo Bientinesi and Robert van de Geijn. "Formal Correctness and Stability of Linear Algebra Algorithms." IMACS05, to appear.
5. Thierry Joffrain, Enrique S. Quintana-Orti, and Robert A. van de Geijn. "Rapid Development of High-Performance Out-of-Core Solvers for Electromagnetics." Proceedings of PARA04, to appear.
6. Paolo Bientinesi, Kazushige Goto, Tze Meng Low, Enrique Quintana-Orti, Robert van de Geijn, and Field Van Zee. "Towards the Final Generation of Dense Linear Algebra Libraries." SC05, submitted.

Technical Reports

1. Tze Meng Low and Robert van de Geijn. "An API for Manipulating Matrices Stored by Blocks." FLAME Working Note #12, UTCS Technical Report TR-2004-15. May 2004.
2. Additional technical reports can be found at `http://www.cs.utexas.edu/users/flame/pubs.html`.

Robert van de Geijn (with G. Rodin (PI), J.C. Browne, and J.T. Oden): Award CMS-9522964 for "Application of High-Performance Computing to Micromechanical Characteristics of Composite Materials." Performance period: 4/95-3/99; Amount: $1,200,000.

The principal results obtained during the project were in the area of **Fast Boundary-Element Methods:** We developed FLEMS, a 3-D boundary-element code for rapid solution of many-particle problems in linear elasticity [13]. This code allows one to integrate any boundary-element method with any iterative strategy that includes a fast summation method. The present version includes the standard collocation boundary-element method and a GMRES solver in which matrix-vector multiplication is performed with the fast multipole method. FLEMS is the first code of its kind for solving multiscale composite materials problems. In addition, advances were made in the areas of fast dislocation dynamics methods [33], modeling error estimates [39, 29], and domain decomposition methods [39, 29, 31].

This grant supported 5 PhD students (all have completed), 1 MS student, and 3 post-doctoral students. In addition, the project supported a visiting scientist from People's Republic of China.

References

[1] M. Barnett, S. Gupta, D. Payne, L. Shuler, R. van de Geijn, and J. Watts. Interprocessor collective communication library. In *Proceedings of Supercomputing 1994*, Nov. 1994.

[2] M. Barnett, S. Gupta, D. Payne, L. Shuler, R. A. van de Geijn, and J. Watts. Interprocessor collective communication library (intercom). In *Proceedings of the Scalable High Performance Computing Conference 1994*, 1994.

[3] M. Barnett, R. Littlefield, D. Payne, and R. van de Geijn. On the efficiency of global combine algorithms for 2-d meshes with wormhole routing. *JPDC*, 24:191–201, 1995.

[4] M. Barnett, D. Payne, and R. van de Geijn. Optimal broadcasting in mesh-connected architectures. Computer Science report TR-91-38, Univ. of Texas, 1991.

[5] Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Trans. Math. Soft.*, 31(1):1–26, March 2005.

[6] Paolo Bientinesi, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. Representing linear algebra algorithms in code: The FLAME APIs. *ACM Trans. Math. Soft.*, 31(1):27–59, March 2005.

[7] J. Bilmes, K. Asanovic, C.W. Chin, and J. Demmel. Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology. In *Proceedings of the International Conference on Supercomputing*. ACM SIGARC, July 1997.

[8] D. Burger, S.W. Keckler, K.S. McKinley, M. Dahlin, L.K. John, Calvin Lin, C.R. Moore, J. Burrill, R.G. McDonald, and W. Yoder. Scaling to the end of silicon with EDGE architectures. *IEEE Computer*, 37(7):44–55, July 2004.

[9] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker. Scalapack: A scalable linear algebra library for distributed memory concurrent computers. In *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 120–127. IEEE Comput. Soc. Press, 1992.

[10] Almadena Chtchelkanova, John Gunnels, Greg Morrow, James Overfelt, and Robert A. van de Geijn. Parallel implementation of BLAS: General techniques for level 3 BLAS. *Concurrency: Practice and Experience*, 9(9):837–857, Sept. 1997.

[11] Jim Demmel and Jack Dongarra. LAPACK 2005 prospectus: Reliable and scalable software for linear algebra computations on high end computers. LAPACK Working Note 164 UT-CS-05-546, University of Tennessee, February 2005.

[12] Jack Dongarra, Robert van de Geijn, and David Walker. Scalability issues affecting the design of a dense linear algebra library. *J. Parallel Distrib. Comput.*, 22(3), Sept. 1994.

[13] Y. Fu, K. J. Klimkowski, G. J. Rodin, E. Berger, J. C. Browne, J. K. Singer, R. A. van de Geijn, and K. S. Vemaganti. A fast solution method for three-dimensional many-particle problems of linear elasticity. *Int. J. Num. Meth. Engrg.*, 42:1215–1229, 1998.

[14] Kazushige Goto. `http://www.cs.utexas.edu/users/kgoto`, 2004.

[15] Kazushige Goto and Robert A. van de Geijn. On reducing TLB misses in matrix multiplication. Technical Report CS-TR-02-55, Department of Computer Sciences, The University of Texas at Austin, 2002.

[16] John Gunnels, Calvin Lin, Greg Morrow, and Robert van de Geijn. A flexible class of parallel matrix multiplication algorithms. In *Proceedings of First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (1998 IPPS/SPDP '98)*, pages 110–116, 1998.

[17] John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal linear algebra methods environment. *ACM Transactions on Mathematical Software*, 27(4):422–455, December 2001.

[18] John A. Gunnels, Greg M. Henry, and Robert A. van de Geijn. A family of high-performance matrix multiplication algorithms. In Vassil N. Alexandrov, Jack J. Dongarra, Benjoe A. Juliano, René S. Renner, and C.J. Kenneth Tan, editors, *Computational Science - ICCS 2001, Part I*, Lecture Notes in Computer Science 2073, pages 51–60. Springer-Verlag, 2001.

[19] John A. Gunnels and Robert A. van de Geijn. Formal methods for high-performance linear algebra libraries. In Ronald F. Boisvert and Ping Tak Peter Tang, editors, *The Architecture of Scientific Software*, pages 193–210. Kluwer Academic Press, 2001.

[20] B. A. Hendrickson and D. E. Womble. The torus-wrap mapping for dense matrix calculations on massively parallel computers. *SIAM J. Sci. Stat. Comput.*, 15(5):1201–1226, 1994.

[21] Introduction to OpenMP. `http://www.llnl.gov/computing/tutorials/openMP/`, 2005.

[22] B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS: High performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Soft.*, 24(3):268–302, 1998.

[23] J. G. Lewis, D. G. Payne, and R. A. van de Geijn. Matrix-vector multiplication and conjugate gradient algorithms on distributed memory computers. In *Proceedings of the Scalable High Performance Computing Conference 1994*, 1994.

[24] W. Lichtenstein and S. L. Johnsson. Block-cyclic dense linear algebra. Technical Report TR-04-92, Harvard University, Center for Research in Computing Technology, Jan. 1992.

[25] Tze Meng Low, Kent Milfeld, Robert van de Geijn, and Field Van Zee. Parallelizing FLAME code with OpenMP task queues. *ACM Trans. Math. Soft.*, submitted.

[26] Tze Meng Low and Robert van de Geijn. An API for manipulating matrices stored by blocks. FLAPACK Working Note #12 TR-2004-15, The University of Texas at Austin, Department of Computer Sciences, May 2004.

[27] Tze Meng Low, Robert van de Geijn, and Field Van Zee. Extracting SMP parallelism for dense linear algebra algorithms from high-level specifications. In *PPoPP'05*, 2005.

[28] Prasenjit Mitra, David Payne, Lance Shuler, Robert van de Geijn, and Jerrell Watts. Fast collective communication libraries, please. In *Proceedings of the Intel Supercomputing Users' Group Meeting 1995*, 1995.

[29] J.T. Oden and T. Zohdi. Analysis and adaptive modeling of highly heterogeneous elastic structures. *Computer Methods in Applied Mechanics and Engineering*, 148:367–391, 1997.

[30] OpenMP Architecture Review Board. `http://www.openmp.org/`, 2005.

[31] E. G. Podnos, I Babuska, and G.J. Rodin. Application of fictitious domain method to micromechanics of non-linear composite materials. *Comp. Meth. in Appl. Mech. Engrg.*, submitted.

[32] POSIX Threads Programming. `http://www.llnl.gov/computing/tutorials/pthreads/`, 2005.

[33] G.J. Rodin. Toward rapid evaluation of the elastic interactions among three-dimensional dislocations. *Phil. Mag. Letters*, 77(4):187–190, 1998.

[34] Sanjiv Shah, Grant Haab, Paul Peterson, and Joe Throop. Flexible control structures for parallelism in OpenMP. In *EWOMP*, 1999.

[35] G. W. Stewart. Communication and matrix computations on large message passing systems. *Parallel Computing*, 16:27–40, 1990.

[36] Ernesto Su, Xinmin Tian, Milind Girkar, Grant Haab, Sanjiv Shah, and Paul Peterson. Compiler support of the workqueuing execution model for Intel SMP architectures. In *EWOMP*, 2002.

[37] Robert A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. The MIT Press, 1997.

[38] R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. In *Proceedings of SC'98*, 1998.

[39] T. I. Zohdi, J. Tinsley Oden, and G.J. Rodin. Hierarchical modeling of heterogeneous bodies. *Computer Methods in Applied Mechanics and Engineering*, 138:273–298, 1996.