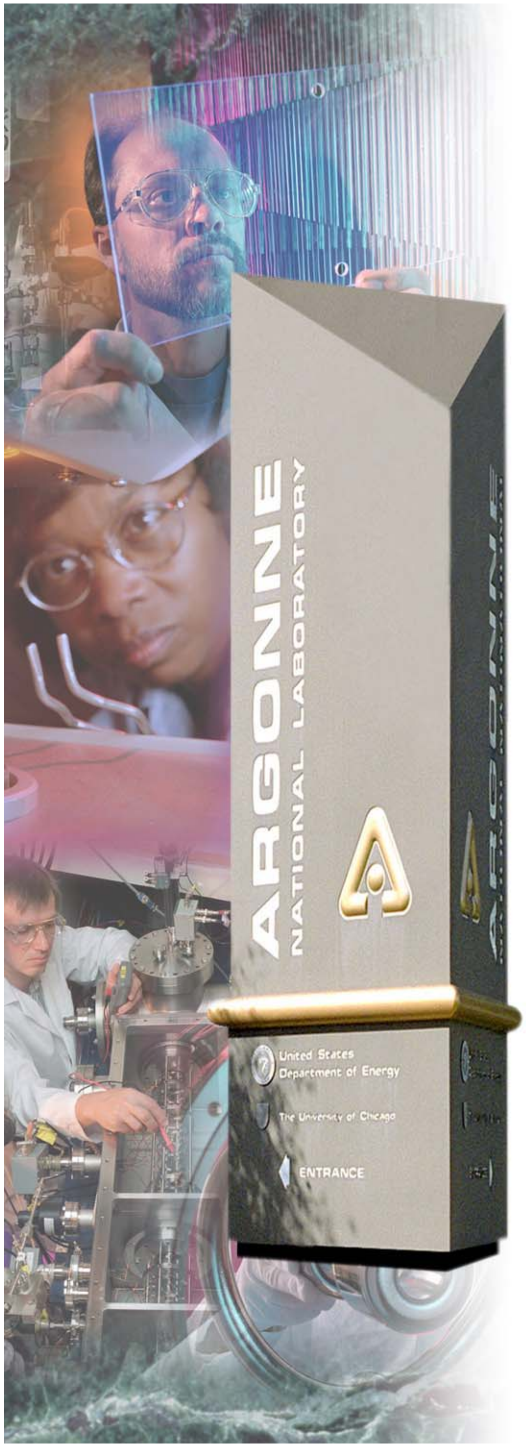


Overcoming the Barriers to Sustained Petaflop Performance

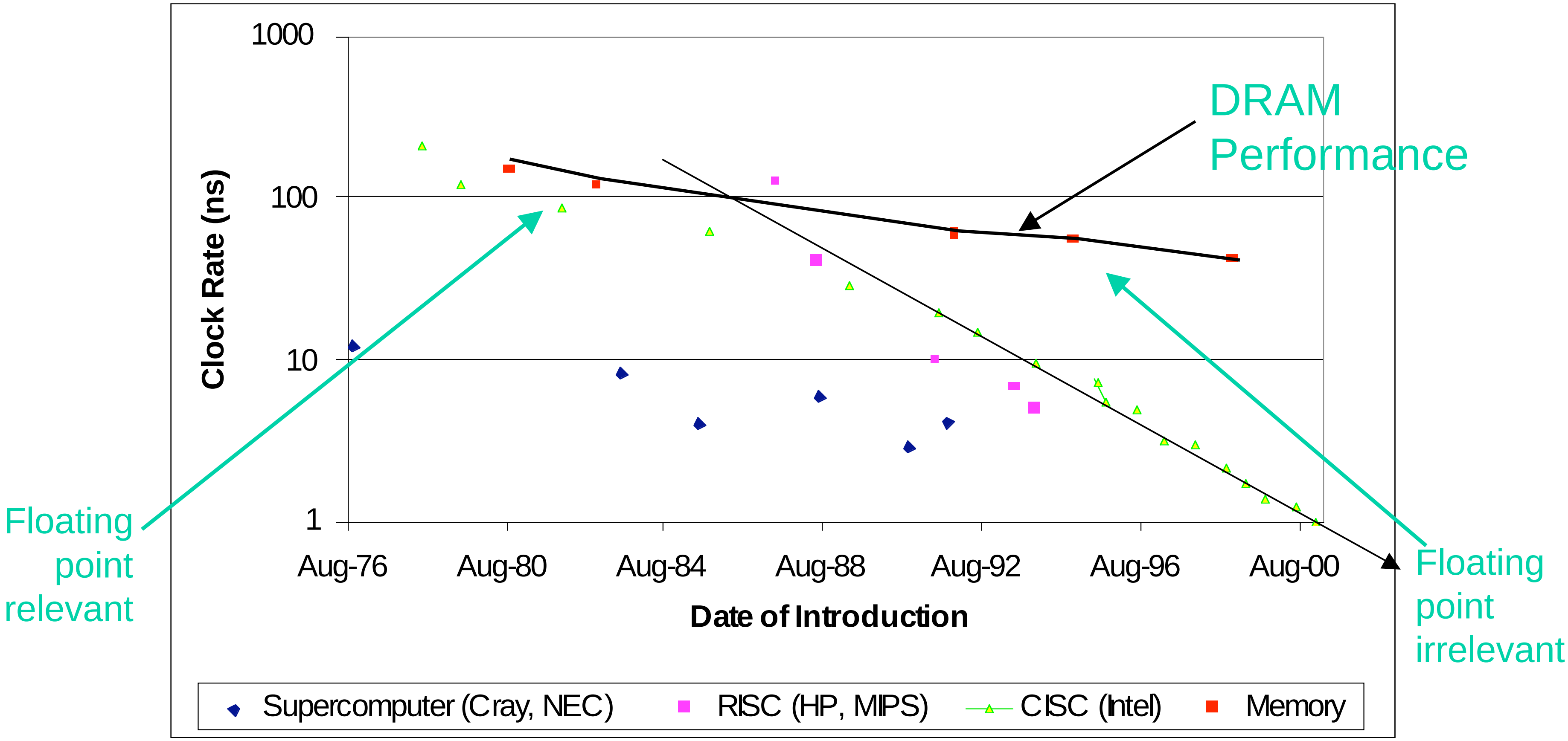
William D. Gropp
Mathematics and Computer Science
www.mcs.anl.gov/~gropp

Argonne National Laboratory

Office of Science
U.S. Department of Energy

A U.S. Department of Energy
Office of Science Laboratory
Operated by The University of Chicago

Why is achieved performance on a single node so poor?



DRAM Performance

Floating point relevant

Floating point irrelevant

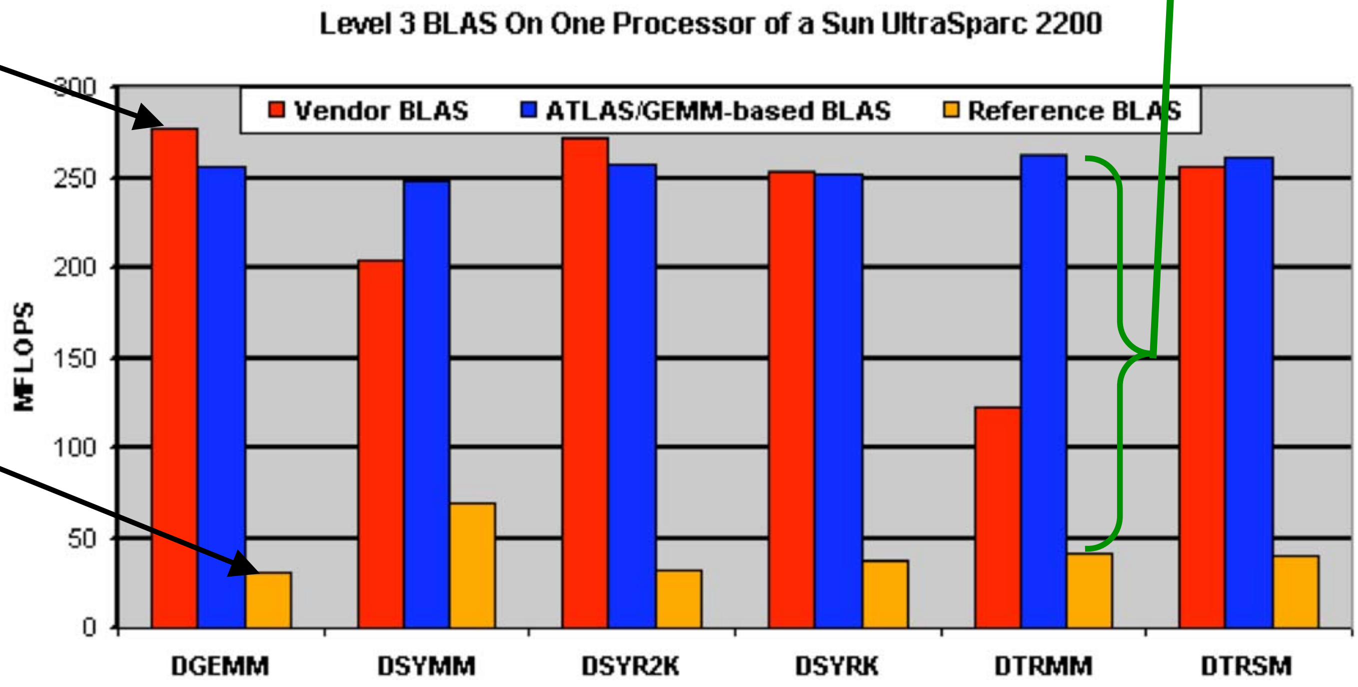
Consequences of Memory/CPU Performance Gap

- **Performance of an application may be (and often is) limited by memory bandwidth or latency rather than CPU clock**
- **“Peak” performance determined by the resource that is operating at full speed for the algorithm**
  - Often memory system (e.g., see STREAM results)
  - Sometimes instruction rate/mix (including integer ops)
- **For example, sparse matrix-vector operation performance is best estimated by using STREAM performance**
  - Note that STREAM performance is delivered performance to a Fortran or C program, not memory bus rate time width
  - High latency of memory and low number of outstanding loads can significantly reduce sustained memory bandwidth

What About CPU-Bound Operations?

- **Dense Matrix-Matrix Product**
  - Probably the numerical program most studied by compiler writers
  - Core of some important applications
  - More importantly, the core operation in High Performance Linpack (HPL)
  - Should give optimal performance…

How Successful are Compilers with CPU Intensive Code?



Large gap between
natural code and
specialized code

Hand-tuned

Compiler

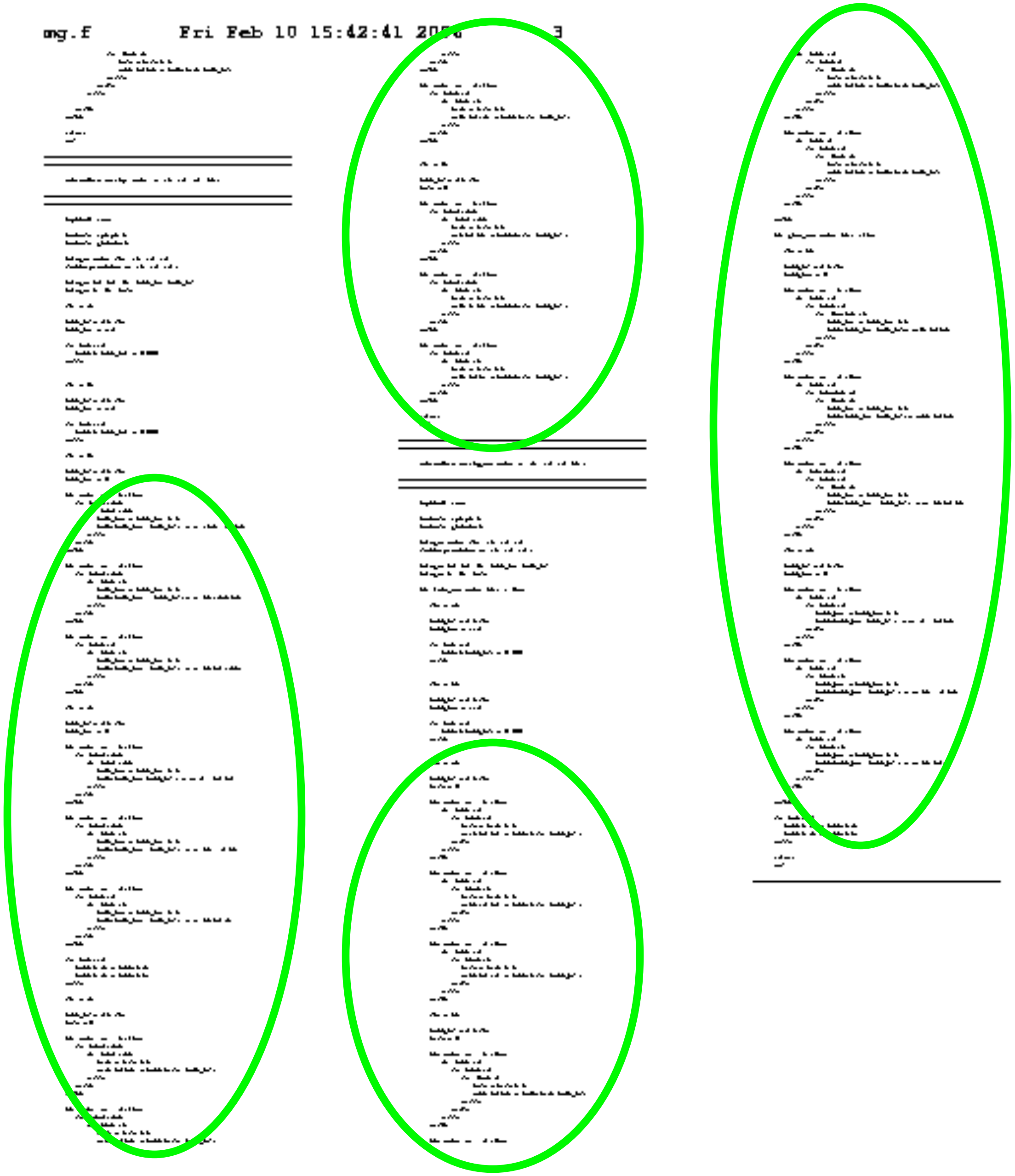
From Atlas

Enormous effort required to get good performance

Distributed Memory code

- **Single node performance is clearly a problem.**
- **What about parallel performance?**
  - Many successes at scale (e.g., Gordon Bell Prizes for >100TF on 64K BG nodes), David’s talk
  - Some difficulties with load-balancing, designing code and algorithms for latency, but skilled programmers and applications scientists have been remarkably successful
- **Is there a problem?**
  - There is the issue of productivity. Consider the NAS parallel benchmark code for Multigrid (mg.f):

mg.f Fri Feb 10 15:42:41 2006 7

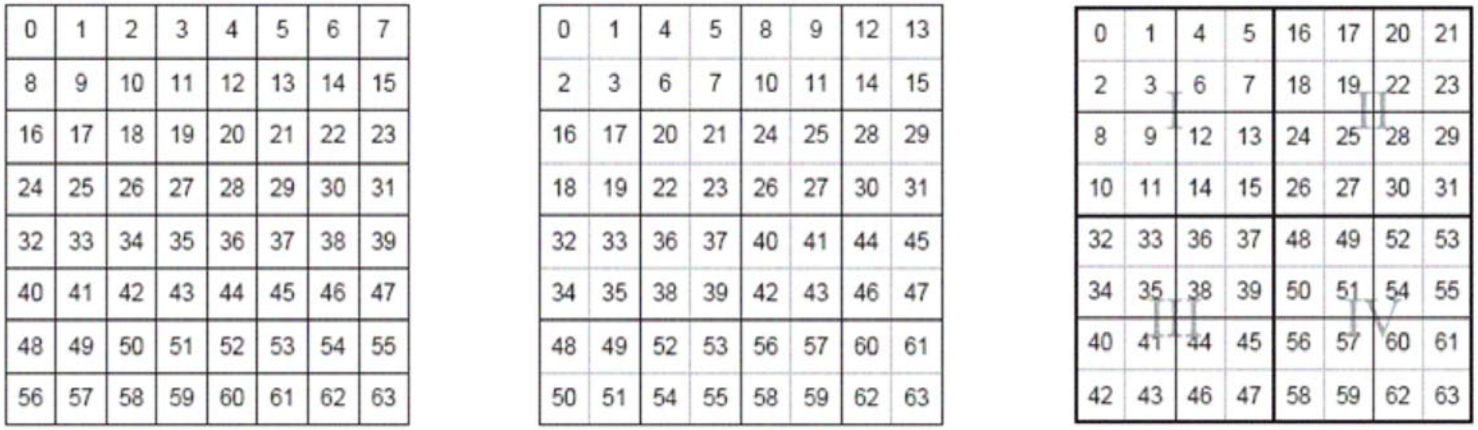


What is the problem?
The user is responsible for all steps in the decomposition of the data structures across the processors

Note that this does give the user (or someone) a great deal of flexibility, as the data structure can be distributed in arbitrary ways across arbitrary sets of processors

Another example…

Manual Decomposition of Data Structures



0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

0	1	4	5	8	9	12	13
2	3	6	7	10	11	14	15
16	17	20	21	24	25	28	29
18	19	22	23	26	27	30	31
32	33	36	37	40	41	44	45
34	35	38	39	42	43	46	47
48	49	52	53	56	57	60	61
50	51	54	55	58	59	62	63

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

- **Trick!**
  - This is from a paper on dense matrix tiling for uniprocessors!
- **This suggests that managing data decompositions is a common problem for real machines, whether they are parallel or not**
  - *Not just an artifact of MPI-style programming*
  - Aiding programmers in data structure decomposition is an important part of solving the productivity puzzle

Possible solutions

- **Single, integrated system**
  - Best choice when it works
    - *Matlab*
    - *Commander Data*
- **Current Terascale systems and many proposed petascale systems exploit hierarchy**
  - Successful at many levels
    - *Cluster hardware*
    - *OS scalability*
  - We should apply this to productivity software
    - *The problem is hard*
    - *Apply classic and very successful Computer Science strategies to address the complexity of generating fast code for a wide range of user-defined data structures.*
- **How can we apply hierarchies?**
  - One approach is to use libraries
    - *Limited by the operations envisioned by the library designer*
  - Another is to enhance the users ability to express the problem in source code

Annotations

- **Aid in the introduction of hierarchy into the software**
  - Its going to happen anyway, so make a virtue of it
- **Create a “market” or ecosystem in transformation tools**
- **Longer term issues**
  - Integrate annotation language into “host” language to ensure type safety, ensure consistency (both syntactic and semantic), closer debugger integration, additional optimization opportunities through information sharing, …

Examples of the Challenges

- **Fast code for DGEMM (dense matrix-matrix multiply)**
  - Code generated by ATLAS omitted to avoid blindness ☺
  - Example code from “Superscalar GEMM-based Level 3 BLAS”, Gustavson et al on the next slide
- **PETSc code for sparse matrix operations**
  - Includes unrolling and use of registers
  - Code for diagonal format is fast on cache-based systems but slow on vector systems.
    - *Too much code to rewrite by hand for new architectures*
- **MPI implementation of collective communication and computation**
  - Complex algorithms for such simple operations as broadcast and reduce are far beyond a compiler’s ability to create from simple code

A fast DGEMM (sample)

```
      SUBROUTINE DGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,
     $                   BETA, C, LDC )
...
                  UISEC = ISEC-MOD( ISEC, 4 )
                  DO 390 J = JJ, JJ+UJSEC-1, 4
                     DO 360 I = II, II+UISEC-1, 4
                        F11 = DELTA*C( I,J )
                        F21 = DELTA*C( I+1,J )
                        F12 = DELTA*C( I,J+1 )
                        F22 = DELTA*C( I+1,J+1 )
                        F13 = DELTA*C( I,J+2 )
                        F23 = DELTA*C( I+1,J+2 )
                        F14 = DELTA*C( I,J+3 )
                        F24 = DELTA*C( I+1,J+3 )
                        F31 = DELTA*C( I+2,J )
                        F41 = DELTA*C( I+3,J )
                        F32 = DELTA*C( I+2,J+1 )
                        F42 = DELTA*C( I+3,J+1 )
                        F33 = DELTA*C( I+2,J+2 )
                        F43 = DELTA*C( I+3,J+2 )
                        F34 = DELTA*C( I+2,J+3 )
                        F44 = DELTA*C( I+3,J+3 )
                        DO 350 L = LL, LL+LSEC-1
                           F11 = F11 + T1( L-LL+1, I-II+1 )*
     $                                              T2( L-LL+1, J-JJ+1 )
                           F21 = F21 + T1( L-LL+1, I-II+2 )*
     $                                              T2( L-LL+1, J-JJ+1 )
                           F12 = F12 + T1( L-LL+1, I-II+1 )*
     $                                              T2( L-LL+1, J-JJ+2 )
                           F22 = F22 + T1( L-LL+1, I-II+2 )*
     $                                              T2( L-LL+1, J-JJ+2 )
                           F13 = F13 + T1( L-LL+1, I-II+1 )*
     $                                              T2( L-LL+1, J-JJ+3 )
                           F23 = F23 + T1( L-LL+1, I-II+2 )*
     $                                              T2( L-LL+1, J-JJ+3 )
                           F14 = F14 + T1( L-LL+1, I-II+1 )*
     $                                              T2( L-LL+1, J-JJ+4 )
                           F24 = F24 + T1( L-LL+1, I-II+2 )*
     $                                              T2( L-LL+1, J-JJ+4 )
                           F31 = F31 + T1( L-LL+1, I-II+3 )*
     $                                              T2( L-LL+1, J-JJ+1 )
                           F41 = F41 + T1( L-LL+1, I-II+4 )*
     $                                              T2( L-LL+1, J-JJ+1 )
                           F32 = F32 + T1( L-LL+1, I-II+3 )*
     $                                              T2( L-LL+1, J-JJ+2 )
                           F42 = F42 + T1( L-LL+1, I-II+4 )*
     $                                              T2( L-LL+1, J-JJ+2 )
                           F33 = F33 + T1( L-LL+1, I-II+3 )*
     $                                              T2( L-LL+1, J-JJ+3 )
                           F43 = F43 + T1( L-LL+1, I-II+4 )*
     $                                              T2( L-LL+1, J-JJ+3 )
                           F34 = F34 + T1( L-LL+1, I-II+3 )*
     $                                              T2( L-LL+1, J-JJ+4 )
                           F44 = F44 + T1( L-LL+1, I-II+4 )*
     $                                              T2( L-LL+1, J-JJ+4 )
  350                   CONTINUE
 ...
*     End of DGEMM.
*
      END
```

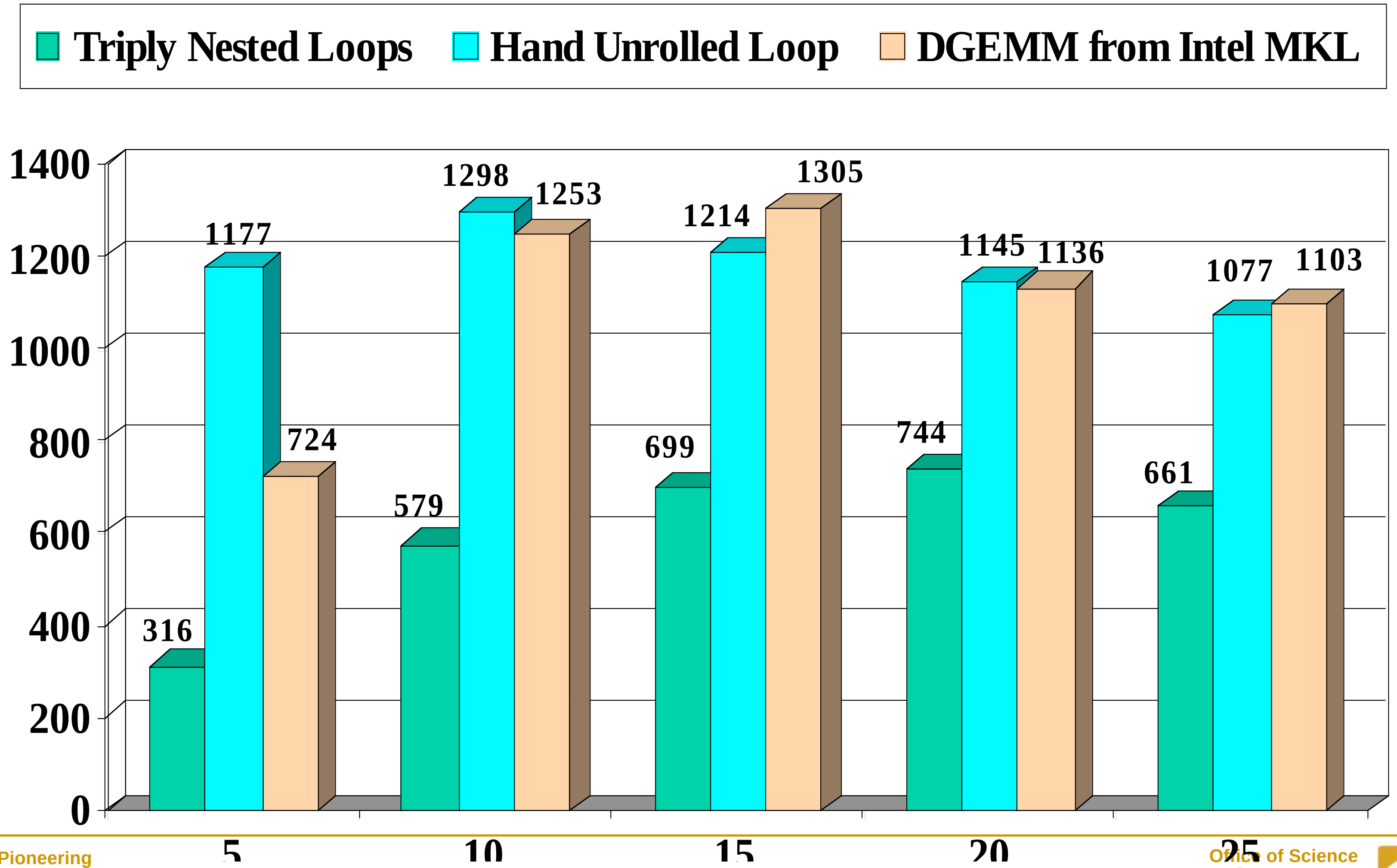
Why not just

```
do i=1,n
   do j=1,m
      c(i,j) = 0
      do k=1,p
         c(i,j) = c(i,j) + a(i,k)*b(k,j)
     enddo
  enddo
enddo
```

Note: This is just part of DGEMM!

Performance of Matrix-Matrix Multiplication
*(MFlops/s vs. n2; n1 = n2; n3 = n2*n2)*

Intel Xeon 2.4 GHz, 512 KB L2 Cache, Intel Compilers at –O3 (Version 8.1), February 12, 2006



Potential challenges faced by languages

1. **Time to develop the language.**
2. **Divergence from mainstream compiler and language development.**
3. **Mismatch with application needs.**
4. **Performance.**
5. **Performance portability.**
6. **Concern of application developers about the success of the language.**

- **Understanding these provides insights into potential solutions**
- **Annotations can *complement* language research by using the principle of *separation of concerns***
- **The annotation approach can be applied to *new* languages, as well**

Advantages of annotations

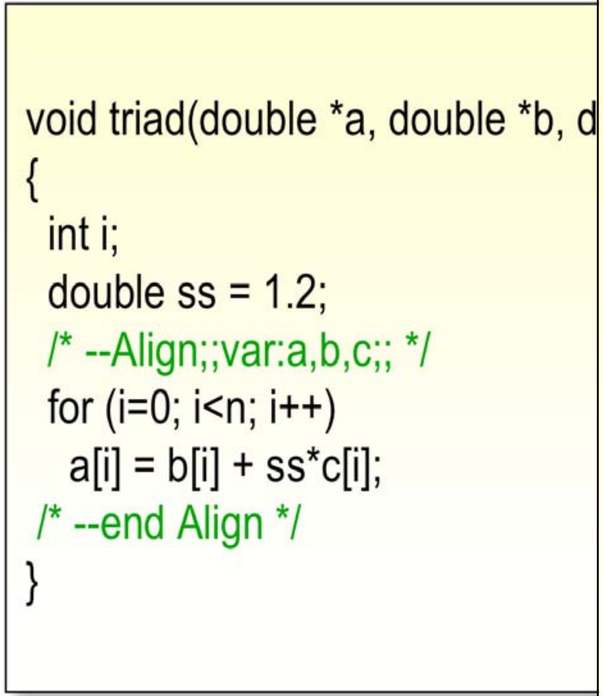
- **These parallel the challenges for languages**

1. **Speeds development and deployment by using source-to-source transformations.**
    - Higher-quality systems can preserve the readability of the source code, avoiding one of the classic drawbacks of preprocessor and source-to-source systems.
2. **Leverages mainstream language developments by building on top of those languages, not replacing them.**
3. **Provides a simpler method to match application needs by allowing experts to develop abstractions tuned to the needs of a class (or even a single important) application.**
    - Also enables the evaluation of new features and data structures

Advantages of annotations (con't)

4. **Provides an effective approach for addressing performance issues by permitting (but not requiring) access by the programmer to low-level details.**
    - Abstractions that allow domain or algorithm-specific approaches to performance can be used because they can be tuned to smaller user communities than is possible in a full language.
5. **Improves performance portability by abstracting platform-specific low-level optimization code.**
6. **Preserves application investment in current languages.**
    - Allows use of existing development tools (debuggers) and allows maintenance and development of code independent of the tools the process the annotations.

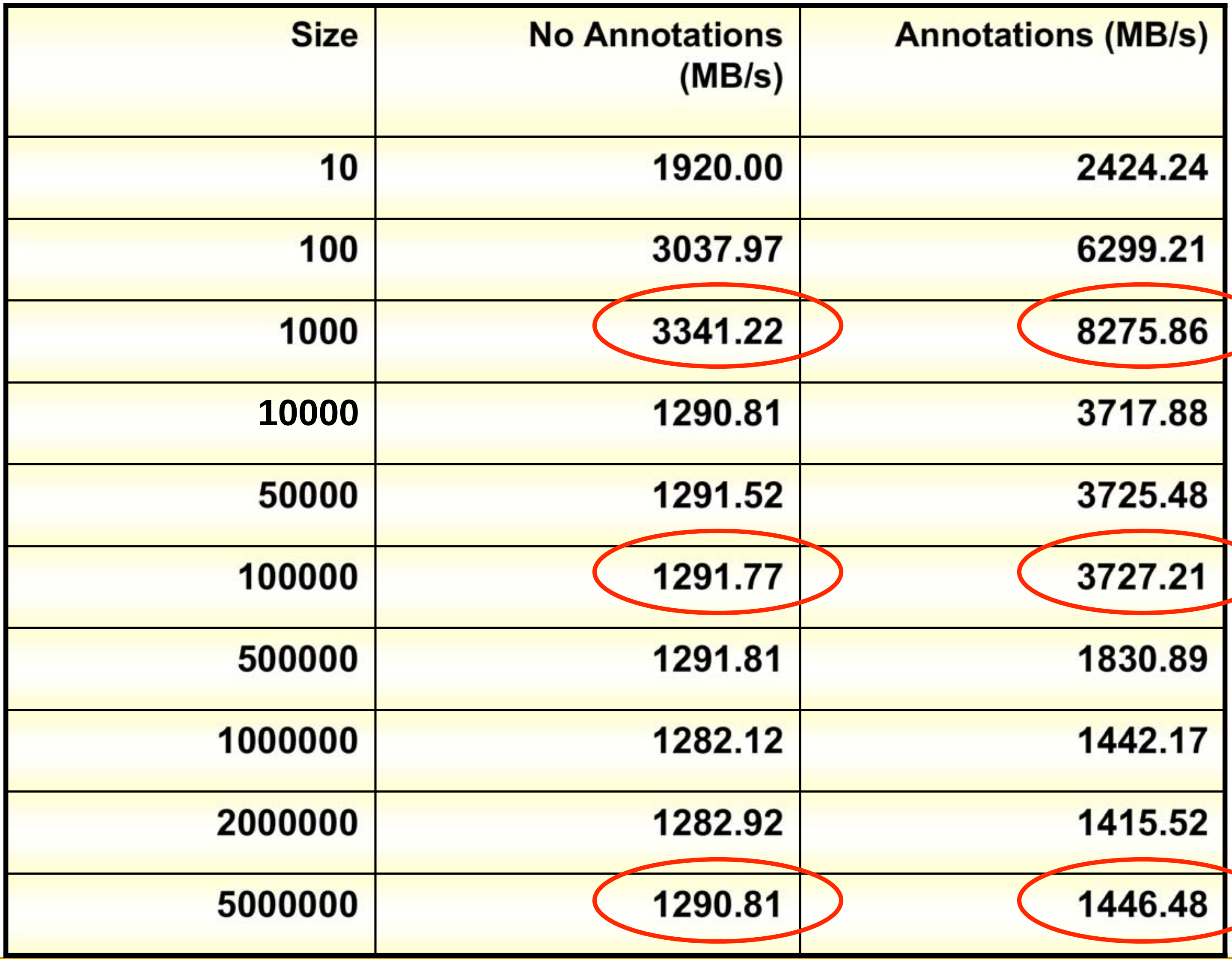
Annotations example: STREAM triad.c for BG/L



```
void triad(double *a, double *b, d
{
  int i;
  double ss = 1.2;
  /* --Align;;var:a,b,c;; */
  for (i=0; i<n; i++)
    a[i] = b[i] + ss*c[i];
 /* --end Align */
}
```

```
void triad(double *a, double *b, double *c, int n)
{
#pragma disjoint (*c,*a,*b)
  int i;
  double ss = 1.2;
  /* --Align;;var:a,b,c;; */
 if ( ((int)(a) | (int)(b) | (int)(c)) & 0xf == 0) {
   __alignx(16,a);
   __alignx(16,b);
   __alignx(16,c);
  for (i=0;i<n;i++) {
    a[i] = b[i] + ss*c[i];
 }
 }
 else {
    for (i=0;i<n;i++) {
        a[i]=b[i] + ss*c[i];
 }
  /* --end Align */
}
```

Simple annotation example: STREAM triad.c on BG/L



Size	No Annotations (MB/s)	Annotations (MB/s)
10	1920.00	2424.24
100	3037.97	6299.21
1000	3341.22	8275.86
10000	1290.81	3717.88
50000	1291.52	3725.48
100000	1291.77	3727.21
500000	1291.81	1830.89
1000000	1282.12	1442.17
2000000	1282.92	1415.52
5000000	1290.81	1446.48

2.5X

2.9X

1.12X

Alternative example: A Regular Mesh Sweep

```
C$AAS Declare Mesh(nx,ny); stencil width 1; a
double precision a(nx,ny)
C$AAE
…
C$AA Init a
…
C$AAS LoopOver a
do i=1, nx
   do j=1, ny
      a(i,j) = a(i-1,j-1) + ….
   enddo
enddo
C$AAE LoopOver nolast
```

Require user provide information on halo (easy for users, hard for compiler)

Regular mesh, distributed across all processes

Hook for initialization

Usual grid sweep, written in “global” coordinates

Generated (Almost Readable!) Code

```
C$AAS Declare Mesh(nx,ny); stencil width 1; a; md5=0xccde2
double precision locala(0:lnx+1,0:lny+1)
C$AAE
…
C$AAS Init a; md5=00
call AAMeshInit(locala,nx,ny,lnx,lny)
C$AAE
…
C$AAS LoopOver a; md5=0xcfd234
call AAMeshExchange(locala,lnx,lny)
do i=1,lnx
    do j=1,lny
        locala(i,j) = locala(i-1,j-1)+…
    enddo
enddo
C$AAE LoopOver nolast
```

Or allocate dynamically

Detect user changes to block

Or explicit MPI-1 or
MPI-2 calls

Or split into Morton
ordered loops

Is This Ugly?

- **You bet!**
  - But it starts the process of considering the code generation process as consisting of a *hierarchy* of solutions
  - Separates the integration of the tools as seen by the user from the integration as seen by “the code”
- **It can evolve toward a cleaner approach, with well-defined interfaces between hierarchies**
- **But only if we accept the need for a hierarchical, compositional approach.**
- **This complements rather than replaces advances in languages, such as global view approaches**

Conclusions

- **It's the memory hierarchy**
- **A pure, compiler based approach is not credible until**
  1. $$\frac{\min(\text{performance of compiler on MM})}{\max(\text{performance of hand-tuned MM})} > 0.9$$
  2. "condition" of that ratio is small (less than 2)
  3. Your favorite performance challenge
- **Compilation is *hard*!**
- **At the node, the memory hierarchy limits performance**
  - Architectural changes can help (e.g., prefetch, more pending loads/stores) but will always need algorithmic and programming help
- **Between nodes, complexity of managing distributed data structures limits productivity, ability to adopt new algorithms**
  - Domain (or better, data-structure) specific nano-languages, used as part of a hierarchical language approach, can help