Domain Specific Abstractions for High-Productivity, High-Performance Scientific Computing

J. (Ram) Ramanujam

Louisiana State Univ.

www.ece.lsu.edu/jxr/

P. (Saday) Sadayappan

The Ohio State Univ.

www.cse.ohio-state.edu/~saday

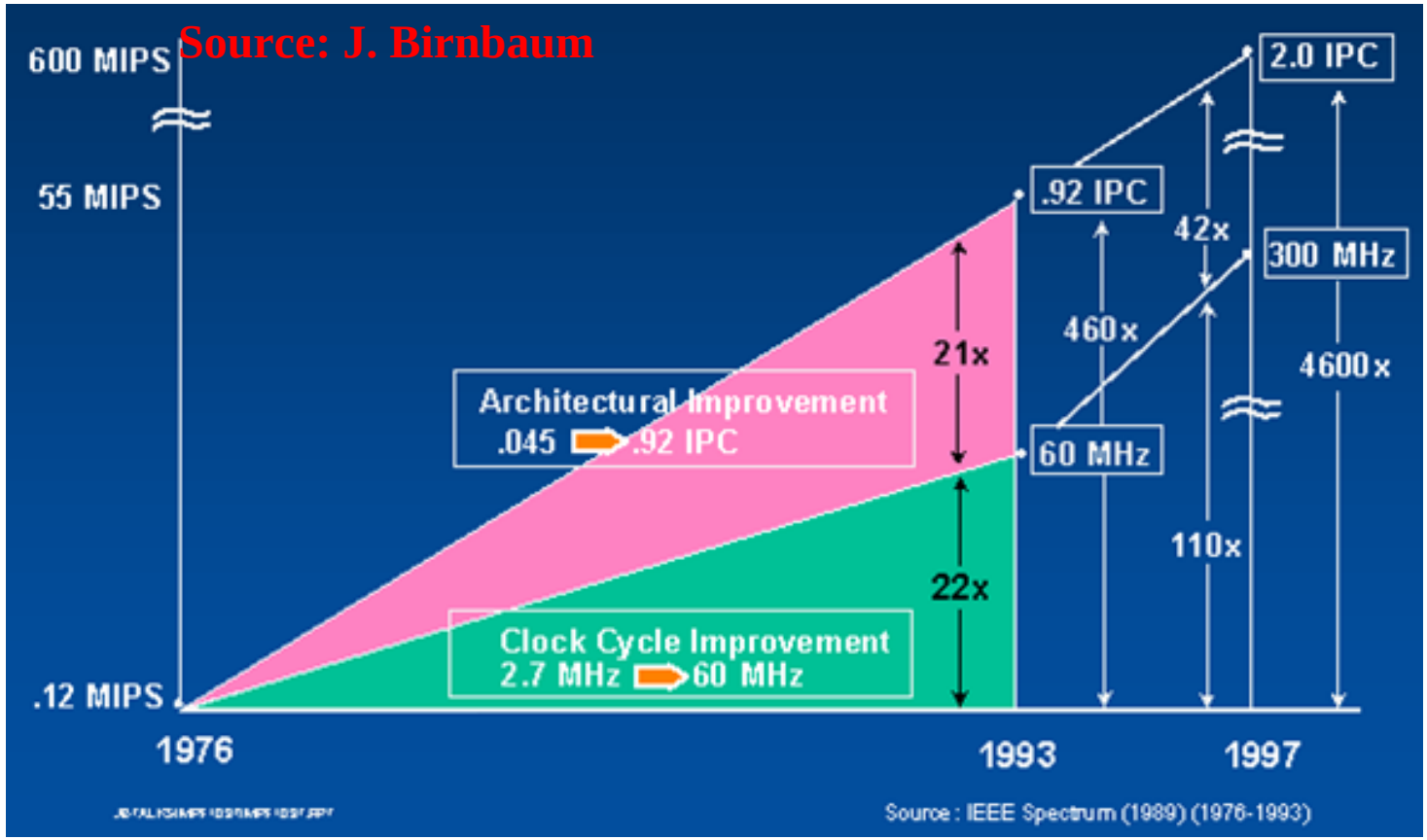
LSU LOUISIANA STATE UNIVERSITY

THE OHIO STATE UNIVERSITY

One-Slide Summary

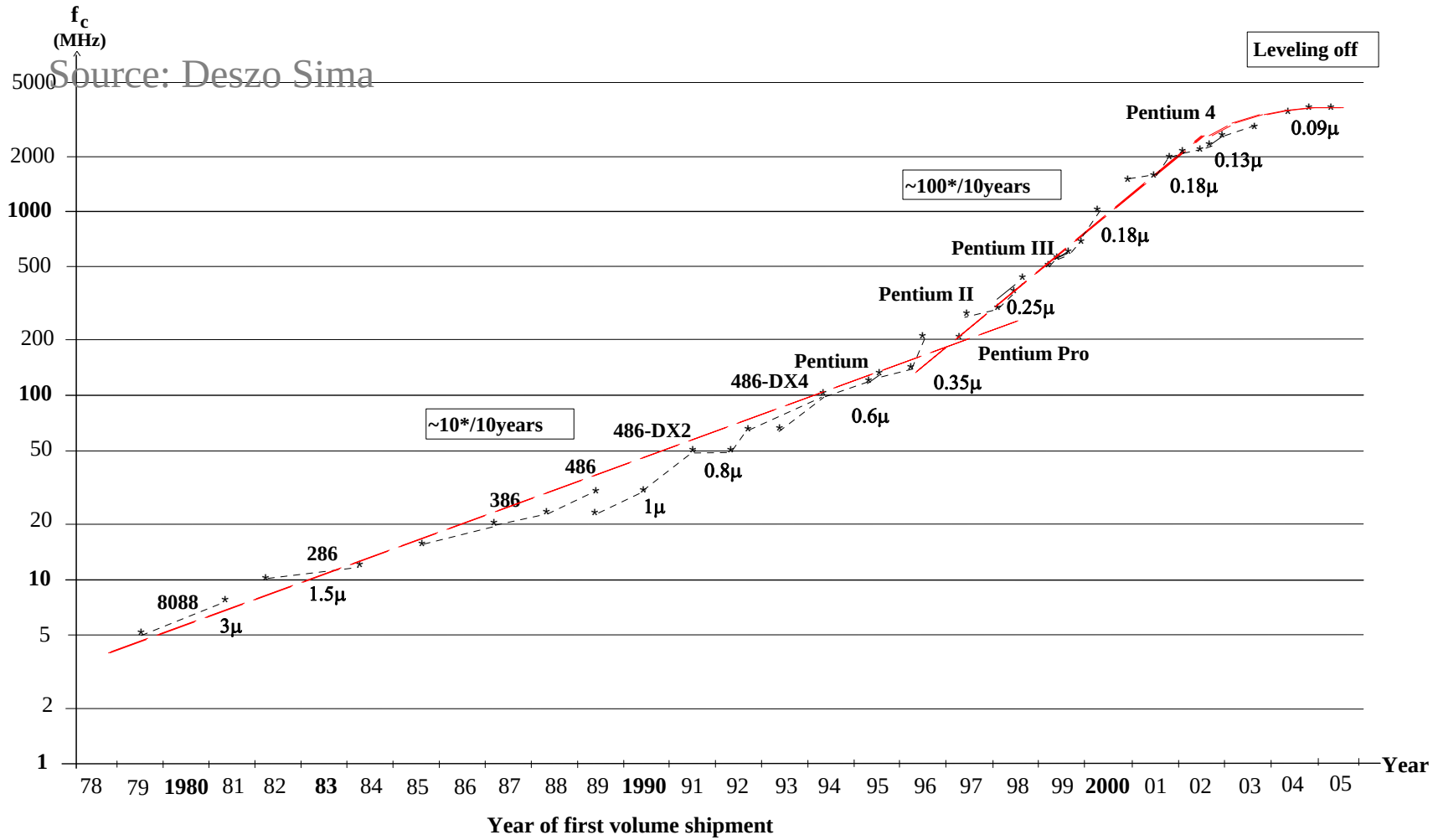
- Multi-core processors have made parallel computing mainstream (not just at National Labs.)
- Developing efficient application software for current/emerging architectures is an increasingly difficult problem
  - Heterogeneity: CPUs and GPUs
- Multi-level layered software abstractions desirable for tensor computations
- Developing effective and scalable domain-specific frameworks requires multi-disciplinary collaboration
- Discuss some of our experiences

The Good Old Days for Software



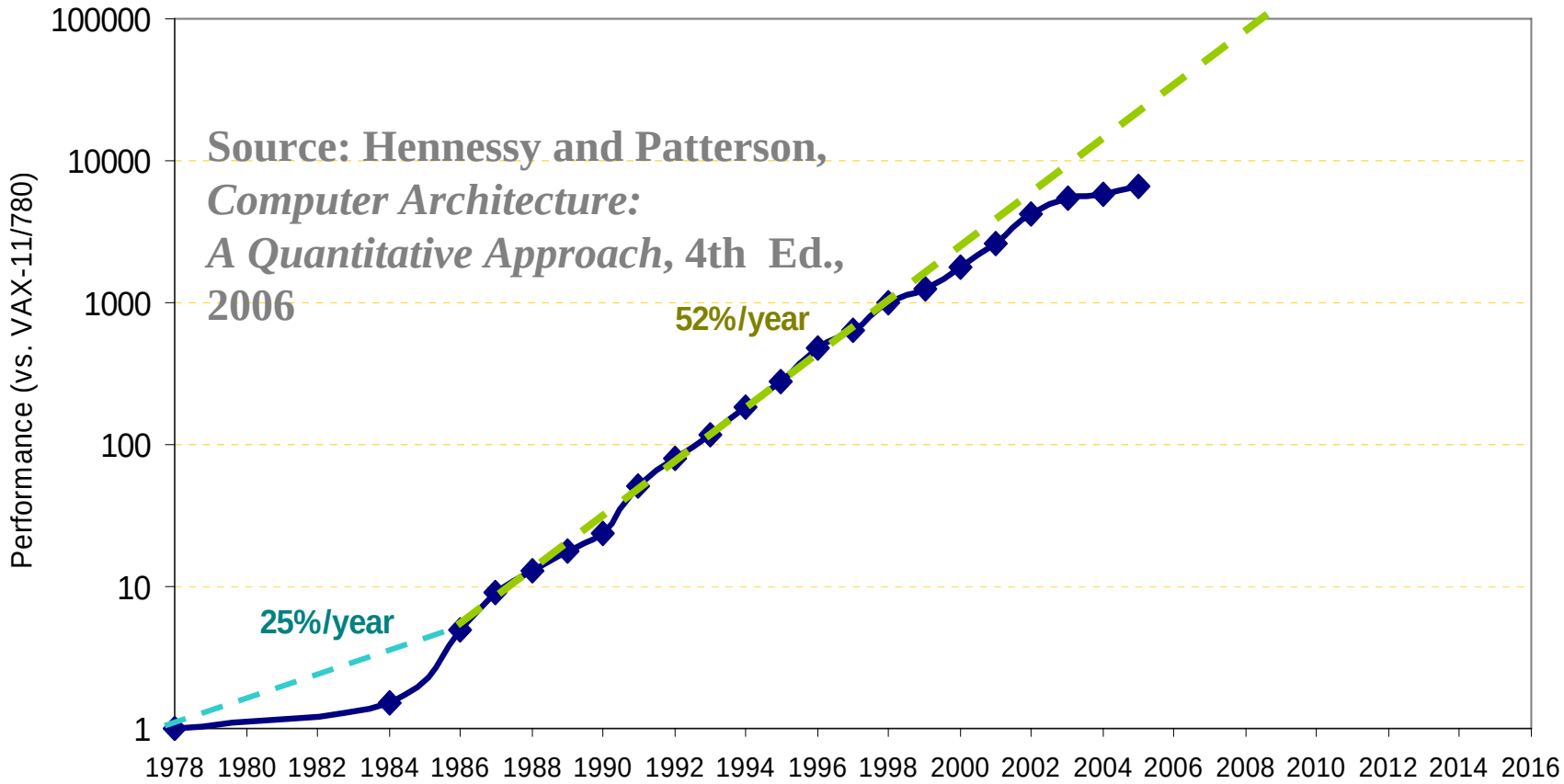
- Single-processor performance experienced dramatic improvements from **clock**, and **architectural** improvement (Pipelining, Instruction-Level-Parallelism)
- Applications experienced automatic performance improvement

Processor Clock Frequency Trends



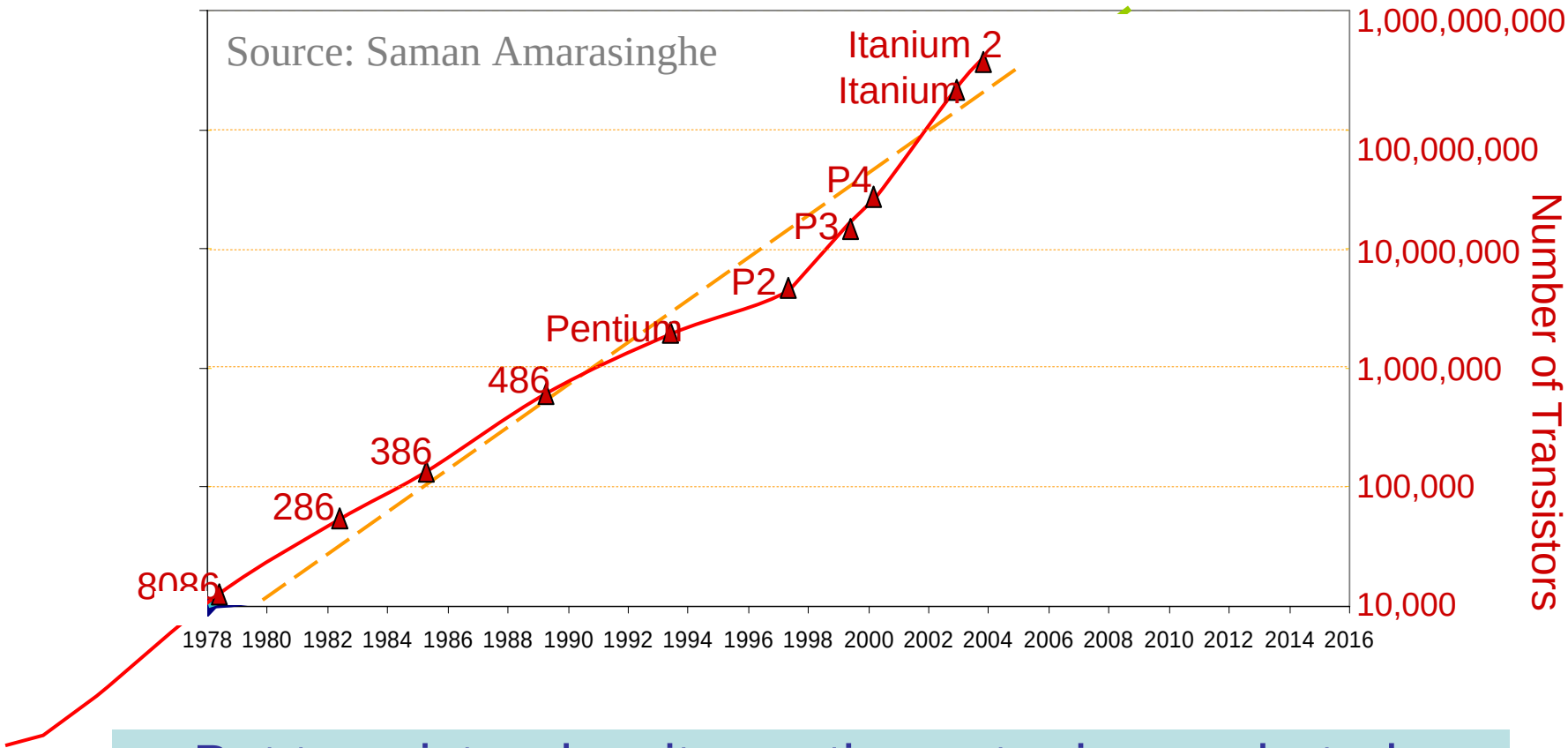
- Processor clock rates increased over 100x in two decades, but have now flattened out

Single-Processor Performance Trends



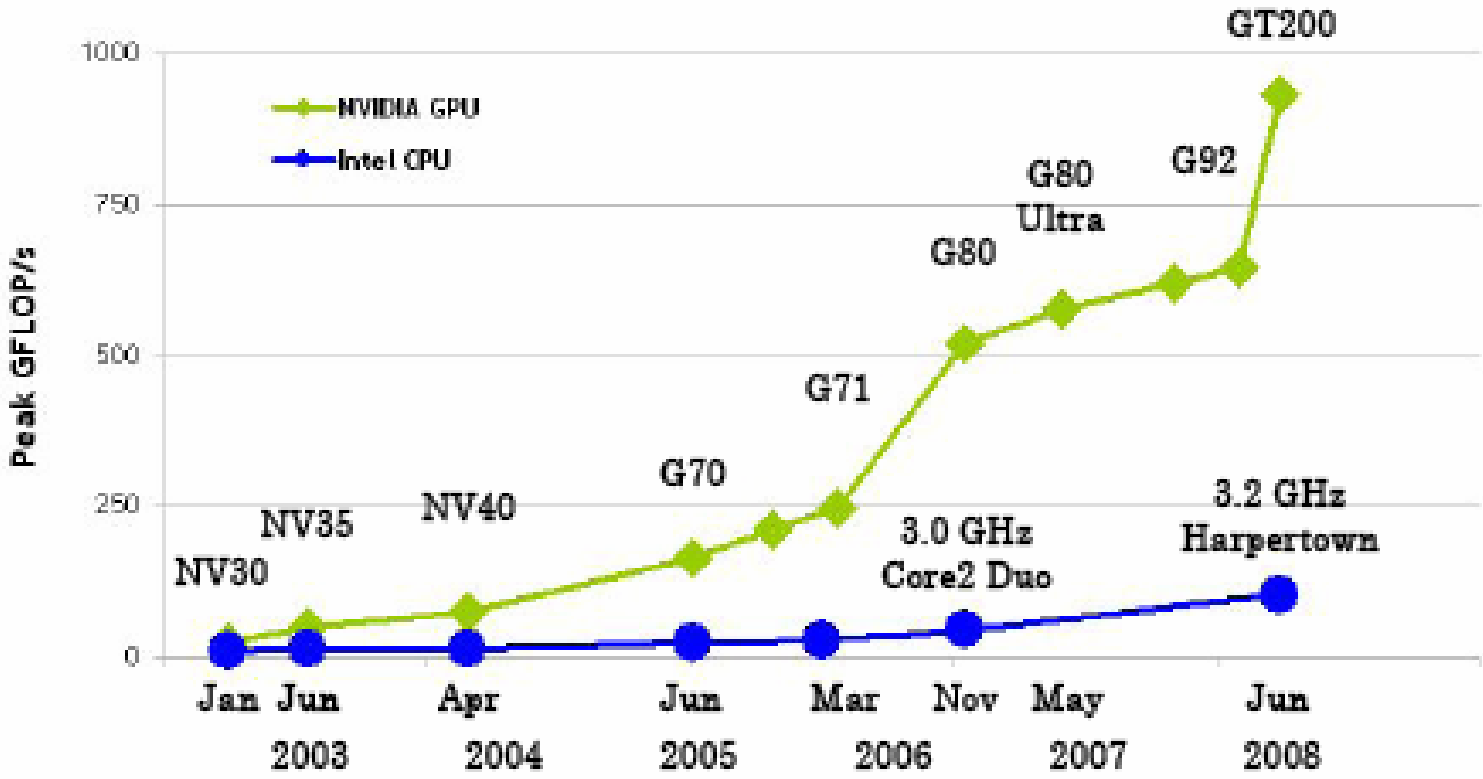
- Single-processor performance improved by ~50%/yr for almost two decades, but has now flattened out: limited further upside from clock or ILP

Moore's Law: Still Going Strong



- But transistor density continues to rise unabated
- Multiple cores are now the best option for sustained performance growth

GPU versus CPU Performance

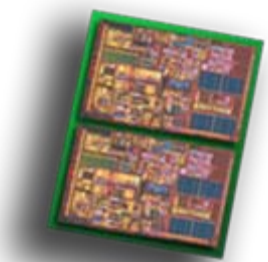


GT200 = GeForce GTX 280	G71 = GeForce 7900 GTX	NV35 = GeForce FX 5950 Ultra
G92 = GeForce 9800 GTX	G70 = GeForce 7800 GTX	NV30 = GeForce FX 5800
G80 = GeForce 8800 GTX	NV40 = GeForce 6800 Ultra	

Source: nVIDIA

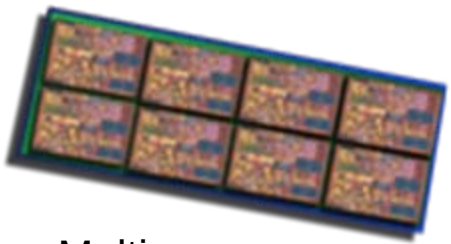
Intel's Vision: Evolutionary Configurable Architecture

Large, Scalar cores for high single-thread performance



Dual core

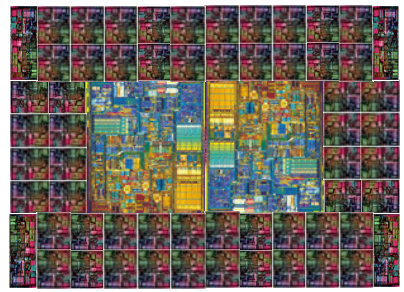
- Symmetric multithreading



Multi-core array

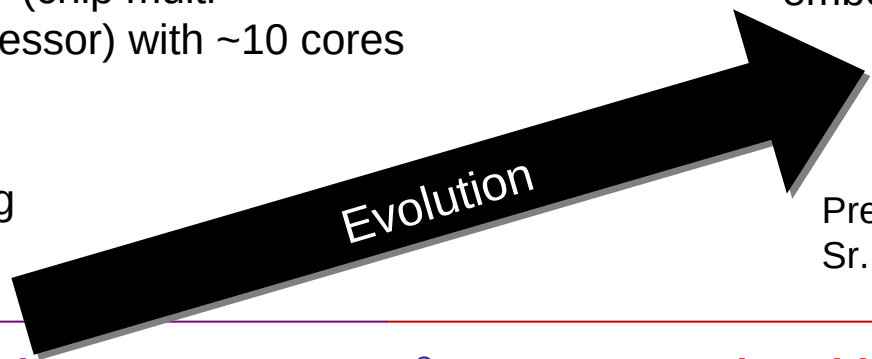
- CMP (chip multi-processor) with ~10 cores

Scalar plus many core for highly threaded workloads



Many-core array

- CMP with 10s-100s low power cores
- Scalar cores
- Capable of TFLOPS+
- Full System-on-Chip
- Servers, workstations, embedded…



Presentation Paul Petersen, Sr. Principal Engineer, Intel

Multi-Core Software Complexity

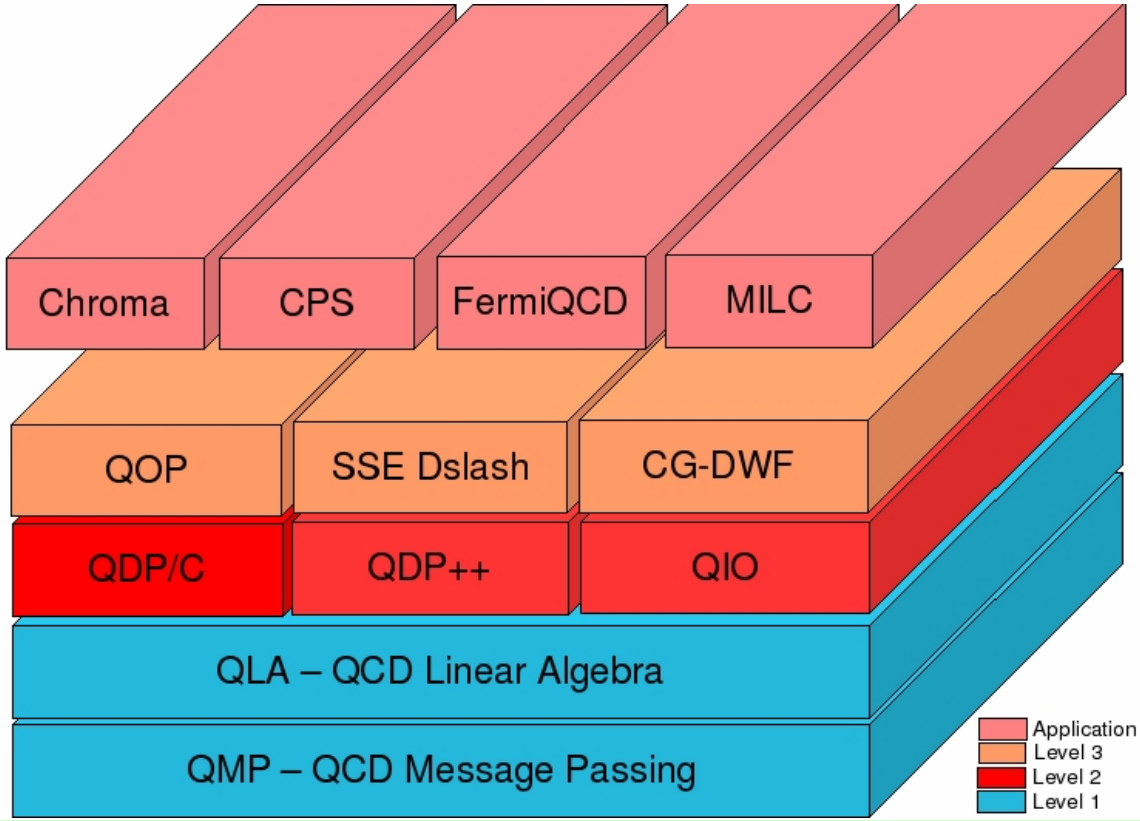
John Hennessy, renown Computer Scientist, President of Stanford, in an interview to ACM Queue in early 2006:

"when we start talking about parallelism and ease of use of truly parallel computers, we're talking about a problem that's as hard as any that computer science has faced. ….I would be panicked if I were in industry."

David Patterson, Berkeley CS Professor, in the same interview:

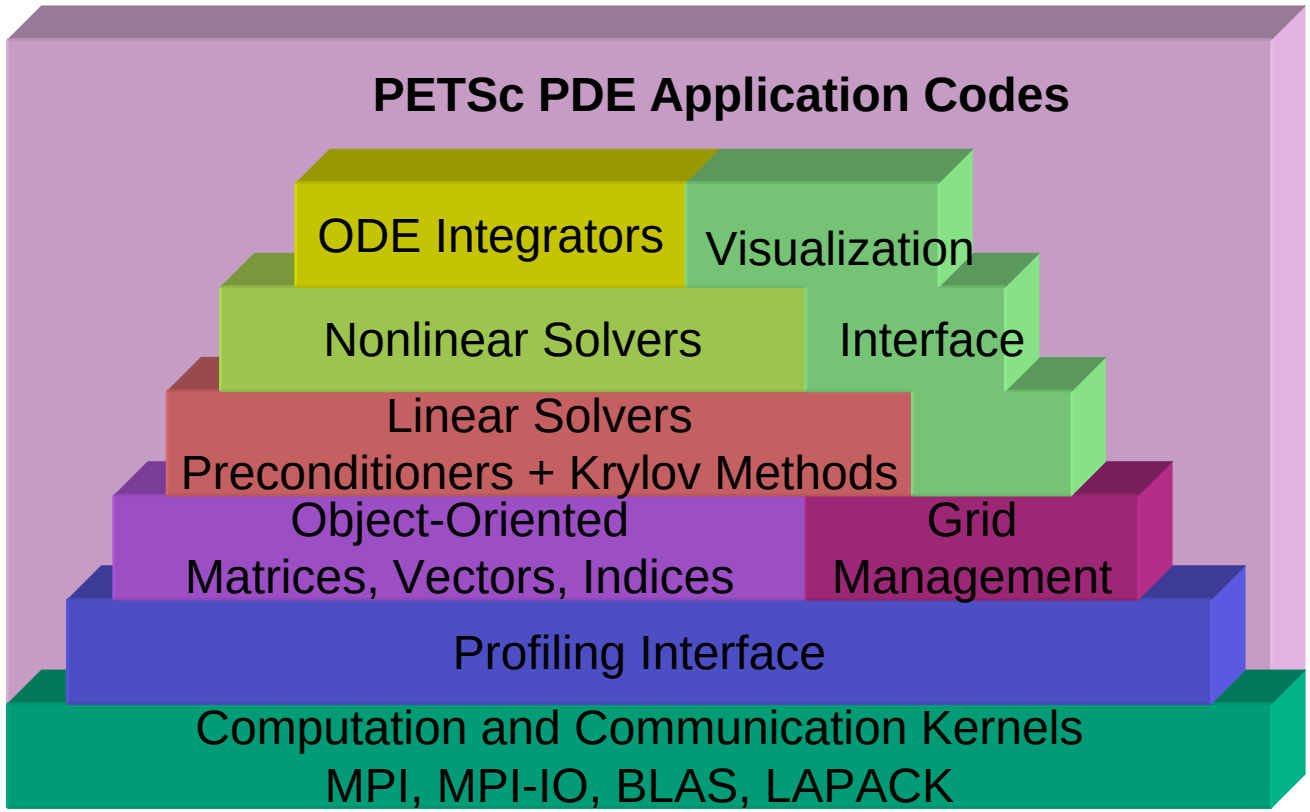
`Parallelism has changed the programming model… At Microsoft in 2005, if you said, "Hey, what do you guys think about parallel computers?" they would reply, "Who cares about parallel computers? We've had 15 or 20 years of doubling every 18 months. Get lost." You couldn't get anybody's attention inside Microsoft by saying that the future was parallelism… In 2006, everybody at Microsoft is talking about parallelism. Five years ago, if you had this breakthrough idea in parallelism, industry would show you the door. Now industry is highly motivated to listen to new ideas.'

Layered API Example: LQCD



- Community-wide collaborative effort to define layered domain-specific API
- Feasible to optimize lower layers for new architectures without redoing applications

Another Layered API: PETSc



- Parallel vectors
  - communicating ghost points
- Parallel matrices
  - several sparse storage formats
  - easy, efficient assembly

- Scalable parallel preconditioners
- Krylov subspace methods
- Parallel Newton nonlinear solvers
- Parallel (ODE) solvers
- Profiling
- Error Checking

Tensor Contraction Engine

Oak Ridge National Laboratory
David E. Bernholdt, *Robert Harrison*

Pacific Northwest National Laboratory
Jarek Nieplocha

Louisiana State University
Gerald Baumgartner ,
J. (Ram) Ramanujam

Ohio State University
Uday Bondhugula, Xiaoyang Gao, Muthu Baskaran, Albert Hartono, Sriram Krishnamoorthy, Quingda Lu, ***Russell Pitzer, P. (Saday) Sadayappan***

University of Florida
So Hirata

University of Waterloo
Marcel Nooijen

Supported by NSF and DOE

Problem Domain: High-Accuracy Quantum Chemical Methods

- Coupled cluster methods are widely used for very high quality electronic structure calculations
- Typical Laplace factorized CCSD(T) term:

Typical methods will have tens to hundreds of such terms

$$A3A = \tfrac{1}{2}\left(X_{ce,af}Y_{ae,cf} + X_{\bar{c}\bar{e},\bar{a}\bar{f}}Y_{\bar{a}\bar{e},\bar{c}\bar{f}} + X_{\bar{c}\bar{e},\bar{a}f}Y_{\bar{a}\bar{e},cf} + X_{\bar{c}\bar{e},a\bar{f}}Y_{\bar{a}e,c\bar{f}} + X_{c\bar{e},\bar{a}\bar{f}}Y_{\bar{a}\bar{e},c\bar{f}} + X_{\bar{c}e,\bar{a}f}Y_{\bar{a}e,cf} + X_{c\bar{e},a\bar{f}}Y_{ae,c\bar{f}}\right)$$

$$X_{ce,af} = t_{ij}^{ce}t_{ij}^{af} \qquad Y_{ae,cf} = \langle ab \| ek \rangle \langle cb \| fk \rangle$$

- Indices i, j, k O(O=100) values, a, b, c, e, f O(V=3000)
- Term costs $O(OV^5) \approx 10^{19}$ FLOPs; Integrals ~ 1000 FLOPs each
- $O(V^4)$ terms ~ 500 TB memory each

The “Tensor Contraction Engine” (TCE)

- User describes computational problem (tensor contractions, a la many-body methods) in a simple, high-level language
  - Similar to what might be written in papers
- Compiler-like tools translate high-level language into traditional Fortran (or C, or…) code
- Generated code is compiled and linked to libraries providing computational infrastructure
  - Code can be tailored to target architecture (e.g. out-of-core version for a PC but in-memory MPI-based code for cluster)
- Two versions of TCE developed
  - Full exploitation of symmetry, but fewer optimizations (So Hirata)
  - Partial exploitation of symmetry, but more sophisticated optimizations
  - Used to implement over 20 models, included in NWChem
  - First parallel implementation for many of the methods

High-Level Language for Tensor Contraction Expressions

```
range V = 3000;
range O = 100;

index a,b,c,d,e,f : V;
index i,j,k : O;

mlimit = 1000000000000;

function F1(V,V,V,O);
function F2(V,V,V,O);

procedure P(in T1[O,O,V,V], in T2[O,O,V,V], out X)=

begin
    X == sum[ sum[F1(a,b,f,k) * F2(c,e,b,k), {b,k}]
                * sum[T1[i,j,a,e] * T2[i,j,c,f], {i,j}],
                {a,e,c,f}];
end
```

$$A3A = \frac{1}{2}(X_{ce,af}Y_{ae,cf} + X_{\bar{c}\bar{e},\bar{a}\bar{f}}Y_{\bar{a}\bar{e},\bar{c}\bar{f}} + X_{\bar{c}\bar{e},\bar{a}\bar{f}}Y_{\bar{a}\bar{e},cf} + X_{\bar{c}\bar{e},\bar{a}\bar{f}}Y_{\bar{a}\bar{e},c\bar{f}} + X_{\bar{c}\bar{e},\bar{a}\bar{f}}Y_{a\bar{e},\bar{c}\bar{f}} + X_{\bar{c}\bar{e},\bar{a}\bar{f}}Y_{a\bar{e},c\bar{f}} + X_{\bar{c}\bar{e},\bar{a}\bar{f}}Y_{\bar{a}\bar{e},\bar{c}\bar{f}})$$

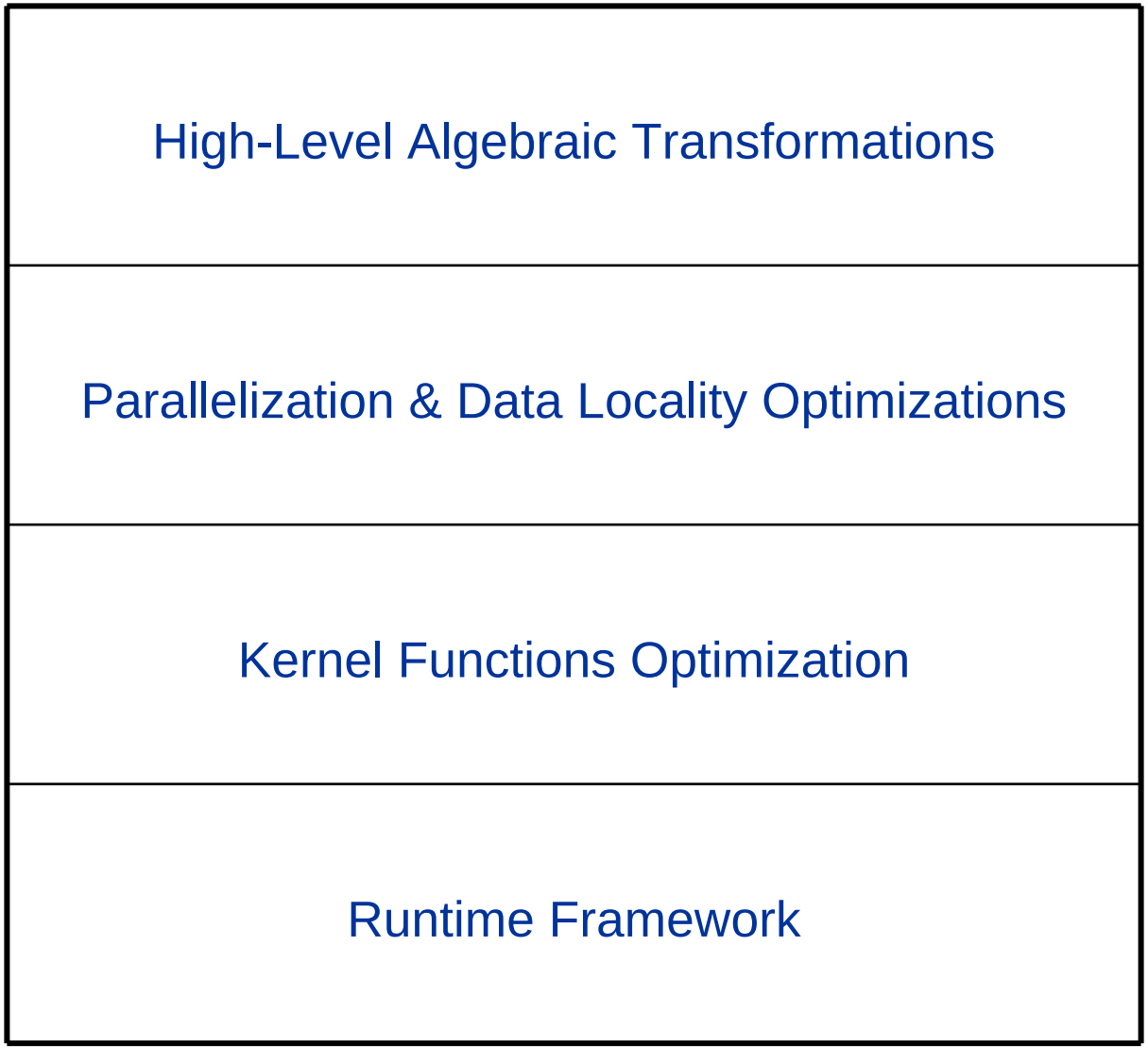
$$X_{ce,af} = t_{ij}^{ce} t_{ij}^{af} \qquad Y_{ae,cf} = \langle ab \| ek \rangle \langle cb \| fk \rangle$$

CCSD Doubles Equation

```
hbar[a,b,i,j] == sum[f[b,c]*t[i,j,a,c],{c}] -sum[f[k,c]*t[k,b]*t[i,j,a,c],{k,c}] +sum[f[a,c]*t[i,j,c,b],{c}] -sum[f[k,c]*t[k,a]*t[i,j,c,b],{k,c}] -sum[f[k,j]*t[i,k,a,b],{k}] -
    sum[f[k,c]*t[j,c]*t[i,k,a,b],{k,c}] -sum[f[k,i]*t[j,k,b,a],{k}] -sum[f[k,c]*t[i,c]*t[j,k,b,a],{k,c}] +sum[t[i,c]*t[j,d]*v[a,b,c,d],{c,d}] +sum[t[i,j,c,d]*v[a,b,c,d],{c,d}]
    +sum[t[j,c]*v[a,b,i,c],{c}] -sum[t[k,b]*v[a,k,i,j],{k}] +sum[t[i,c]*v[b,a,j,c],{c}] -sum[t[k,a]*v[b,k,j,i],{k}] -sum[t[k,d]*t[i,j,c,b]*v[k,a,c,d],{k,c,d}] -
    sum[t[i,c]*t[j,k,b,d]*v[k,a,c,d],{k,c,d}] -sum[t[j,c]*t[k,b]*v[k,a,c,i],{k,c}] +2*sum[t[j,k,b,c]*v[k,a,c,i],{k,c}] -sum[t[j,k,c,b]*v[k,a,c,i],{k,c}] -
    sum[t[i,c]*t[j,d]*t[k,b]*v[k,a,d,c],{k,c,d}] +2*sum[t[k,d]*t[i,j,c,b]*v[k,a,d,c],{k,c,d}] -sum[t[k,b]*t[i,j,c,d]*v[k,a,d,c],{k,c,d}] -sum[t[j,d]*t[i,k,c,b]*v[k,a,d,c],{k,c,d}]
    +2*sum[t[i,c]*t[j,k,b,d]*v[k,a,d,c],{k,c,d}] -sum[t[i,c]*t[j,k,d,b]*v[k,a,d,c],{k,c,d}] -sum[t[j,k,b,c]*v[k,a,i,c],{k,c}] -sum[t[i,c]*t[k,b]*v[k,a,j,c],{k,c}] -
    sum[t[i,k,c,b]*v[k,a,j,c],{k,c}] -sum[t[i,c]*t[j,d]*t[k,a]*v[k,b,c,d],{k,c,d}] -sum[t[k,d]*t[i,j,a,c]*v[k,b,c,d],{k,c,d}] -sum[t[k,a]*t[i,j,c,d]*v[k,b,c,d],{k,c,d}]
    +2*sum[t[j,d]*t[i,k,a,c]*v[k,b,c,d],{k,c,d}] -sum[t[j,d]*t[i,k,c,a]*v[k,b,c,d],{k,c,d}] -sum[t[i,c]*t[j,k,d,a]*v[k,b,c,d],{k,c,d}] -sum[t[i,c]*t[k,a]*v[k,b,c,j],{k,c}]
    +2*sum[t[i,k,a,c]*v[k,b,c,j],{k,c}] -sum[t[i,k,c,a]*v[k,b,c,j],{k,c}] +2*sum[t[k,d]*t[i,j,a,c]*v[k,b,d,c],{k,c,d}] -sum[t[j,d]*t[i,k,a,c]*v[k,b,d,c],{k,c,d}] -
    sum[t[j,c]*t[k,a]*v[k,b,i,c],{k,c}] -sum[t[j,k,c,a]*v[k,b,i,c],{k,c}] -sum[t[i,k,a,c]*v[k,b,j,c],{k,c}] +sum[t[i,c]*t[j,d]*t[k,a]*t[l,b]*v[k,l,c,d],{k,l,c,d}] -
    2*sum[t[k,b]*t[l,d]*t[i,j,a,c]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[k,a]*t[l,d]*t[i,j,c,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[k,a]*t[l,b]*t[i,j,c,d]*v[k,l,c,d],{k,l,c,d}] -
    2*sum[t[j,c]*t[l,d]*t[i,k,a,b]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[j,d]*t[l,b]*t[i,k,a,c]*v[k,l,c,d],{k,l,c,d}] +sum[t[j,d]*t[l,b]*t[i,k,c,a]*v[k,l,c,d],{k,l,c,d}] -
    2*sum[t[i,c]*t[l,d]*t[j,k,b,a]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,a]*t[j,k,b,d]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,b]*t[j,k,d,a]*v[k,l,c,d],{k,l,c,d}]
    +sum[t[i,k,c,d]*t[j,l,b,a]*v[k,l,c,d],{k,l,c,d}] +4*sum[t[i,k,a,c]*t[j,l,b,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,k,c,a]*t[j,l,b,d]*v[k,l,c,d],{k,l,c,d}] -
    2*sum[t[i,k,a,b]*t[j,l,c,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,k,a,c]*t[j,l,d,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,k,c,a]*t[j,l,d,b]*v[k,l,c,d],{k,l,c,d}]
    +sum[t[i,c]*t[j,d]*t[k,l,a,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,j,c,d]*t[k,l,a,b]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,j,c,b]*t[k,l,a,d]*v[k,l,c,d],{k,l,c,d}] -
    2*sum[t[i,j,a,c]*t[k,l,b,d]*v[k,l,c,d],{k,l,c,d}] +sum[t[j,c]*t[k,b]*t[l,a]*v[k,l,c,i],{k,l,c}] +sum[t[l,c]*t[j,k,b,a]*v[k,l,c,i],{k,l,c}] -2*sum[t[l,a]*t[j,k,b,c]*v[k,l,c,i],{k,l,c}]
    +sum[t[l,a]*t[j,k,c,b]*v[k,l,c,i],{k,l,c}] -2*sum[t[k,c]*t[j,l,b,a]*v[k,l,c,i],{k,l,c}] +sum[t[k,a]*t[j,l,b,c]*v[k,l,c,i],{k,l,c}] +sum[t[k,b]*t[j,l,c,a]*v[k,l,c,i],{k,l,c}]
    +sum[t[j,c]*t[l,k,a,b]*v[k,l,c,i],{k,l,c}] +sum[t[i,c]*t[k,a]*t[l,b]*v[k,l,c,j],{k,l,c}] +sum[t[l,c]*t[i,k,a,b]*v[k,l,c,j],{k,l,c}] -2*sum[t[l,b]*t[i,k,a,c]*v[k,l,c,j],{k,l,c}]
    +sum[t[l,b]*t[i,k,c,a]*v[k,l,c,j],{k,l,c}] +sum[t[i,c]*t[k,l,a,b]*v[k,l,c,j],{k,l,c}] +sum[t[j,c]*t[l,d]*t[i,k,a,b]*v[k,l,d,c],{k,l,c,d}]
    +sum[t[j,d]*t[l,b]*t[i,k,a,c]*v[k,l,d,c],{k,l,c,d}] +sum[t[j,d]*t[l,a]*t[i,k,c,b]*v[k,l,d,c],{k,l,c,d}] -2*sum[t[i,k,c,d]*t[j,l,b,a]*v[k,l,d,c],{k,l,c,d}] -
    2*sum[t[i,k,a,c]*t[j,l,b,d]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,c,a]*t[j,l,b,d]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,a,b]*t[j,l,c,d]*v[k,l,d,c],{k,l,c,d}]
    +sum[t[i,k,c,b]*t[j,l,d,a]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,a,c]*t[j,l,d,b]*v[k,l,d,c],{k,l,c,d}] +sum[t[k,a]*t[l,b]*v[k,l,i,j],{k,l}] +sum[t[k,l,a,b]*v[k,l,i,j],{k,l}]
    +sum[t[k,b]*t[l,d]*t[i,j,a,c]*v[l,k,c,d],{k,l,c,d}] +sum[t[k,a]*t[l,d]*t[i,j,c,b]*v[l,k,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,d]*t[j,k,b,a]*v[l,k,c,d],{k,l,c,d}] -
    2*sum[t[i,c]*t[l,a]*t[j,k,b,d]*v[l,k,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,a]*t[j,k,d,b]*v[l,k,c,d],{k,l,c,d}] +sum[t[i,j,c,b]*t[k,l,a,d]*v[l,k,c,d],{k,l,c,d}]
    +sum[t[i,j,a,c]*t[k,l,b,d]*v[l,k,c,d],{k,l,c,d}] -2*sum[t[l,c]*t[i,k,a,b]*v[l,k,c,j],{k,l,c}] +sum[t[l,b]*t[i,k,a,c]*v[l,k,c,j],{k,l,c}] +sum[t[l,a]*t[i,k,c,b]*v[l,k,c,j],{k,l,c}]
    +v[a,b,i,j]
```

In the coupled cluster method with single and double excitations (CCSD) the “singles” and “doubles” equations are iterated until convergence and that solution is used to evaluate the molecular energy

Multi-Level Optimization Framework



Algebraic Transformation: Example

$$S(a,b,i,j) = \sum_{c,d,e,f,k,l} A(a,c,i,k)B(b,e,f,l)C(d,f,j,k)D(c,d,e,l)$$ $4N^{10}$ Ops

$$S(a,b,i,j) = \sum_{c,d,e,f,k,l} A(a,c,i,k)C(d,f,j,k)B(b,e,f,l)D(c,d,e,l)$$

$$S(a,b,i,j) = \sum_{c,d,f,k} A(a,c,i,k)C(d,f,j,k)\left(\sum_{e,l} B(b,e,f,l)D(c,d,e,l)\right)$$

$$S(a,b,i,j) = \sum_{c,k} A(a,c,i,k)\left[\sum_{d,f} C(d,f,j,k)\left(\sum_{e,l} B(b,e,f,l)D(c,d,e,l)\right)\right]$$

$$T1(b,c,d,f) = \sum_{e,l} B(b,e,f,l)D(c,d,e,l)$$ $2N^6$ Ops

$$T2(b,c,j,k) = \sum_{d,f} T1(b,c,d,f)C(d,f,j,k)$$ $2N^6$ Ops

$$S(a,b,i,j) = \sum_{c,k} T2(b,c,j,k)A(a,c,i,k)$$ $2N^6$ Ops

Algebraic Transformation: Summary

$$S(a,b,i,j) = \sum_{c,d,e,f,k,l} A(a,c,i,k)B(b,e,f,l)C(d,f,j,k)D(c,d,e,l)$$

- Requires $4 * N^{10}$ operations if indices a-l have range N
- Optimized form requires only $6 * N^{6}$ operations

$$T1(b,c,d,f) = \sum_{e,l} B(b,e,f,l)D(c,d,e,l)$$

$$T2(b,c,j,k) = \sum_{d,f} T1(b,c,d,f)C(d,f,j,k)$$

$$S(a,b,i,j) = \sum_{c,k} T2(b,c,j,k)A(a,c,i,k)$$

- Optimization Problem: Given an input tensor-contraction expression, find equivalent form that minimizes # operations
  - Problem is NP-hard; efficient pruning search strategy developed, that has been very effective in practice

- However, storage requirements increase after operation minimization

Memory Minimization: Compute by Parts (Loop Fusion)

$$T1_{bcdf} = \sum_{e,l} B_{befl} D_{cdel}$$

$$T2_{bcjk} = \sum_{d,f} T1_{bcdf} C_{dfjk}$$

$$S_{abij} = \sum_{c,k} T2_{bcjk} A_{acik}$$

Formula sequence

```
T1 = 0; T2 = 0; S = 0
for b, c, d, e, f, l
    T1bcdf += Bbefl Dcdel
for b, c, d, f, j, k
    T2bcjk += T1bcdf Cdfjk
for a, b, c, i, j, k
    Sabij += T2bcjk Aacik
```

Unfused code

```
S = 0
for b, c
    T1f = 0; T2f = 0
    for d, e, f, l
        T1fdf += Bbefl Dcdel
    for d, f, j, k
        T2fjk += T1fdf Cdfjk
    for a, i, j, k
        Sabij += T2fjk Aacik
```

(Partially) Fused code

Memory Minimization: Loop Fusion

```
T1 = 0; T2 = 0; S = 0
for b, c, d, e, f, l
    T1[bcdf] += B[befl] D[cdel]
for b, c, d, f, j, k
    T2[bcjk] += T1[bcdf] C[dfjk]
for a, b, c, i, j, k
    S[abij] += T2[bcjk] A[acik]
```

Unfused code

```
S = 0
for b, c
    T1f = 0; T2f = 0
    for d, e, f, l
        T1f[df] += B[befl] D[cdel]
    for d, f, j, k
        T2f[jk] += T1f[df] C[dfjk]
    for a, i, j, k
        S[abij] += T2f[jk] A[acik]
```

(Partially) Fused code

```
S = 0
for b, c
    T1f = 0; T2f = 0
    for d, f
        for e, l
            T1f += B[befl] D[cdel]
        for j, k
            T2f[jk] += T1f C[dfjk]
    for a, i, j, k
        S[abij] += T2f[jk] A[acik]
```

Fully Fused code

- Optimization Problem: Given an operation-minimized sequence of tensor-contractions, find "best" set of loops to fuse, to minimize memory access overhead
  - Problem is NP-hard; heuristics and pruning search used

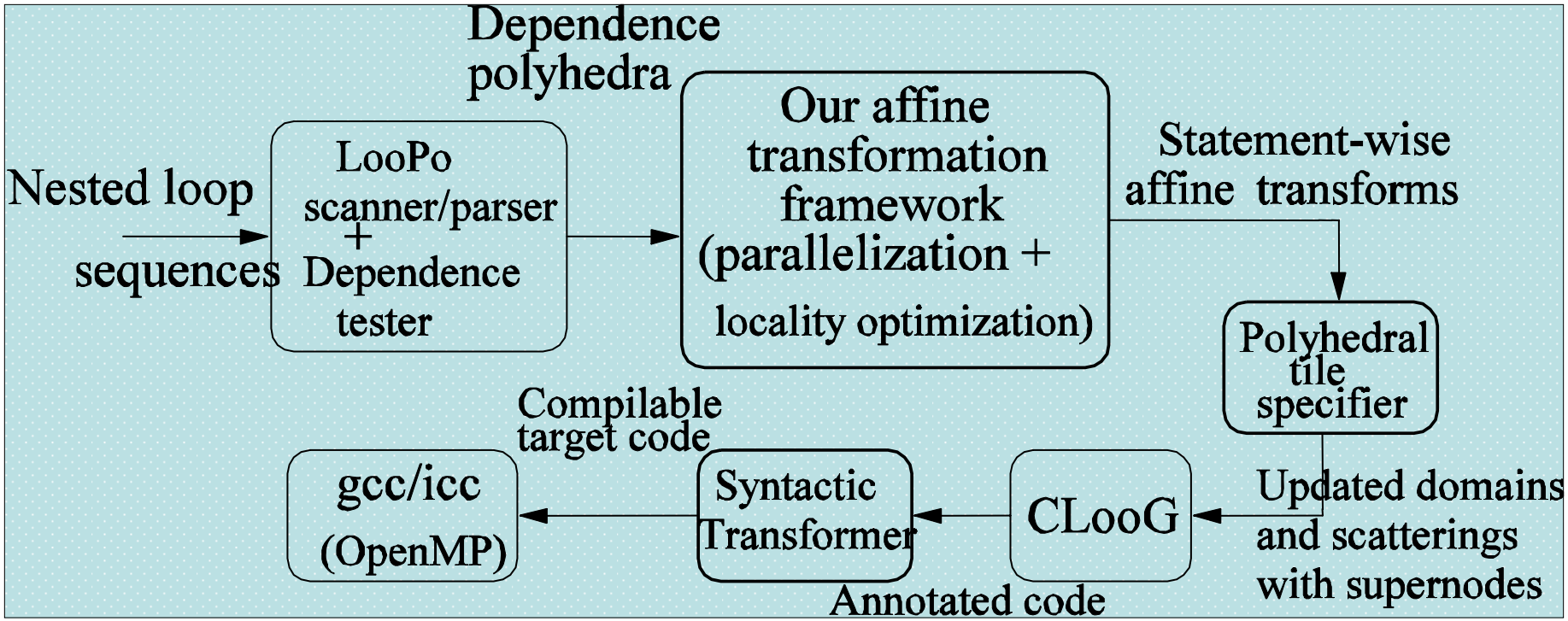
Towards High-Performance Tensor Computations

- Tensor computations expressible as nested loops operating on multi-dimensional arrays
- We see several possible approaches
  - Use a compiler optimization framework to automatically optimize loops with complex nesting structure
  - Develop Basic Tensor Algebra Subroutines (BTAS)
  - Exploit BLAS – How?
- Tensors are not always dense. Here are some challenges
  - Exploiting symmetry
  - Exploiting sparsity
  - Exploiting block-sparsity (RINO: Regular Inner Nonregular Outer computations)

Automatic Optimization of Sequential Code

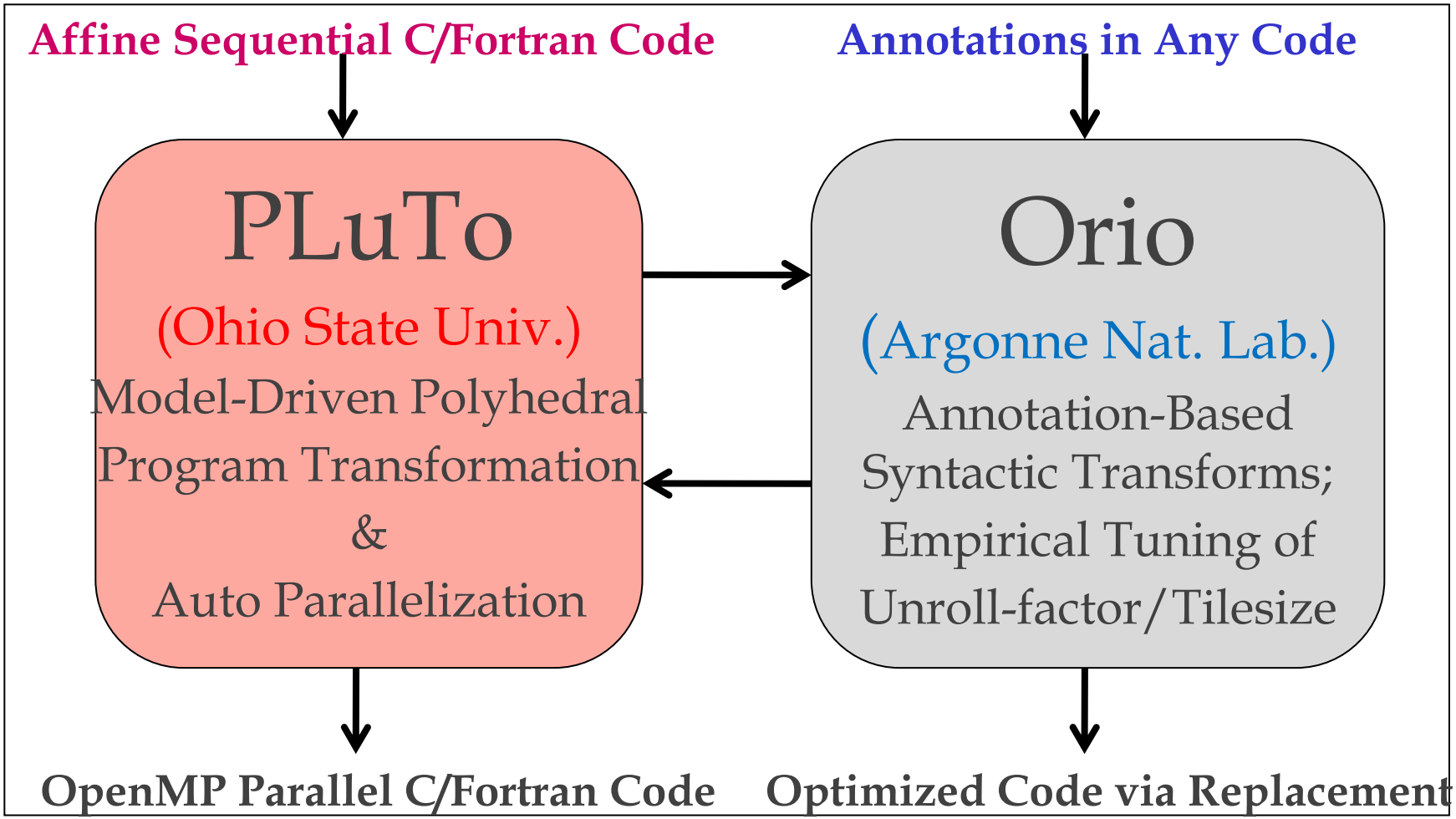
- Automatic optimization and parallelization has been a long-sought goal
  - Large body of compiler optimization research
  - Heightened interest now with ubiquity of multi-core processors
- Several vendor compilers offer an automatic parallelization feature for SMP/multi-core systems
  - Limited use in practice; users do explicit parallelization
  - From ORNL website: “The automatic parallelization performed by the compiler is of limited utility, however. Performance may increase little or may even decrease.”
- Polyhedral compiler framework holds promise
  - Prototype automatic parallelization system for regular (affine) computations: PLuTo

PLuTo Automatic Parallelizer



- Fully automatic transformation of sequential input C or Fortran code (affine) into tiled OpenMP-parallel code
- Available at http://sourceforge.net/projects/pluto-compiler

PLuTo and Orio



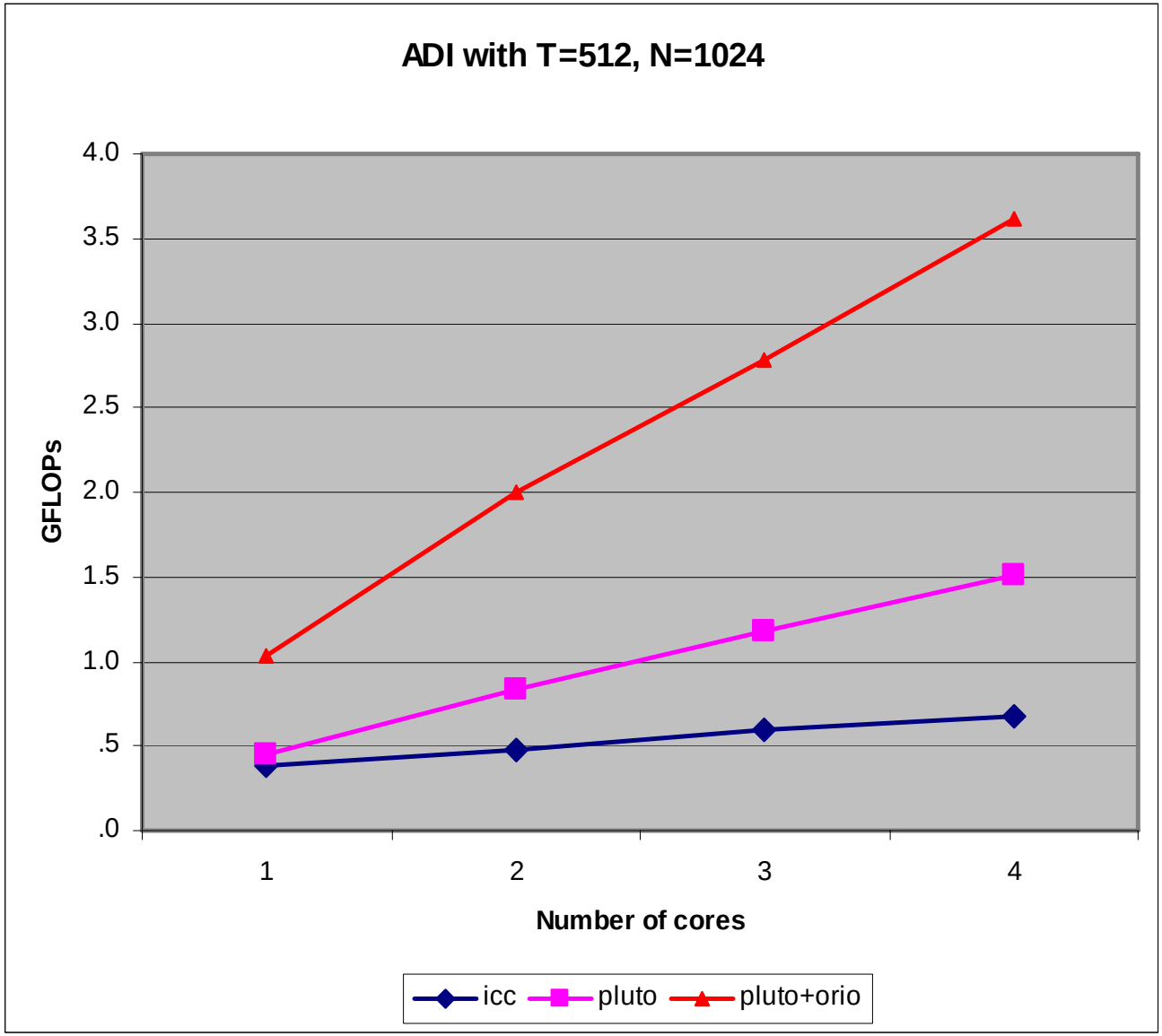
http://sourceforge.net/projects/pluto-compiler

https://trac.mcs.anl.gov/projects/performance/wiki/Orio

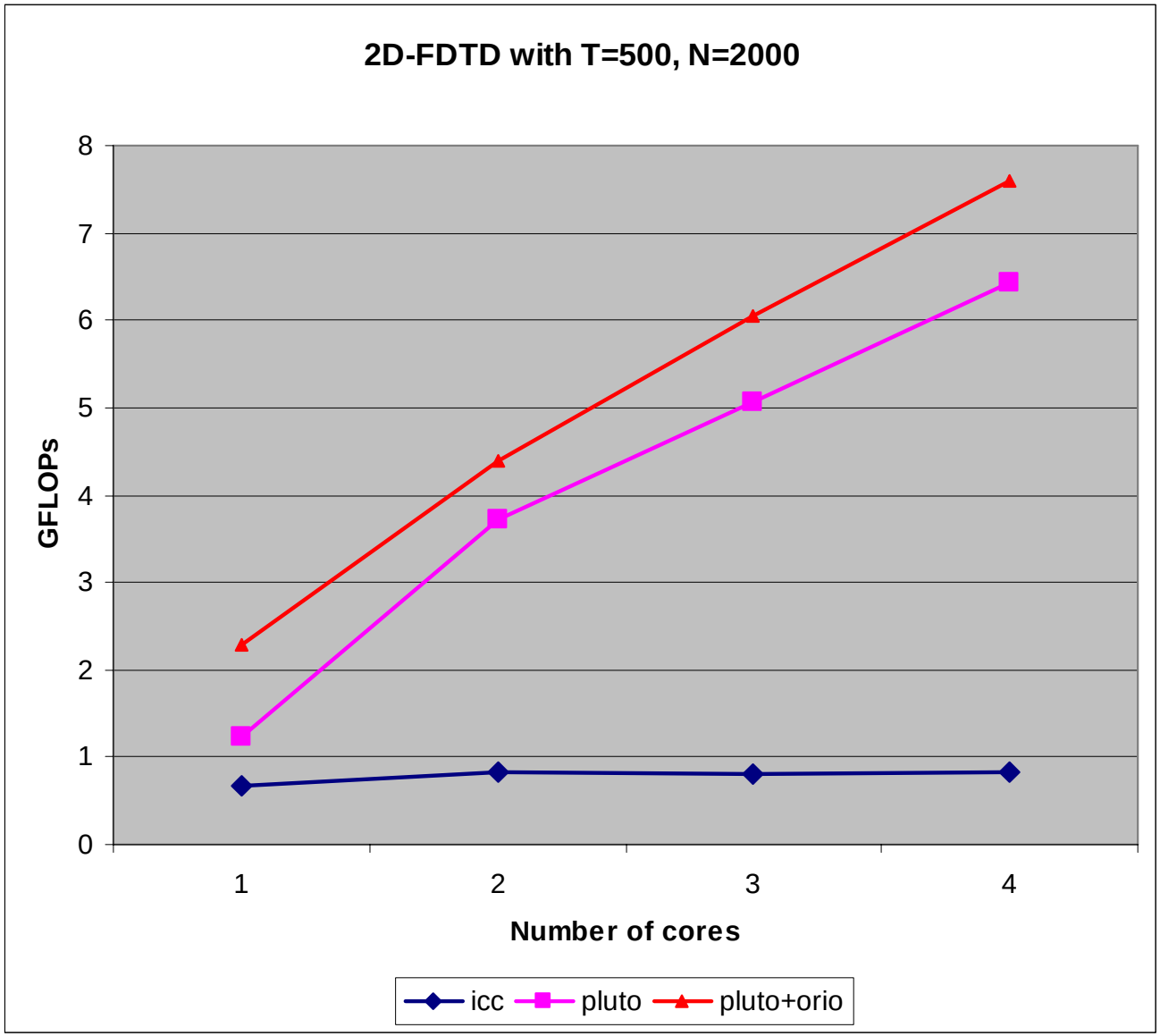
Experimental Results from PLuTo+Orio

- Intel Core2 Quad Q6600 2.4 GHz (quad core with shared L2 cache), FSB 1066 MHz, DDR2 667 RAM
- 32 KB L1 cache, 8 MB L2 cache (4MB per core pair)
- ICC 10.1 (-oparallel -fast)
- Linux 2.6.18 x86-64

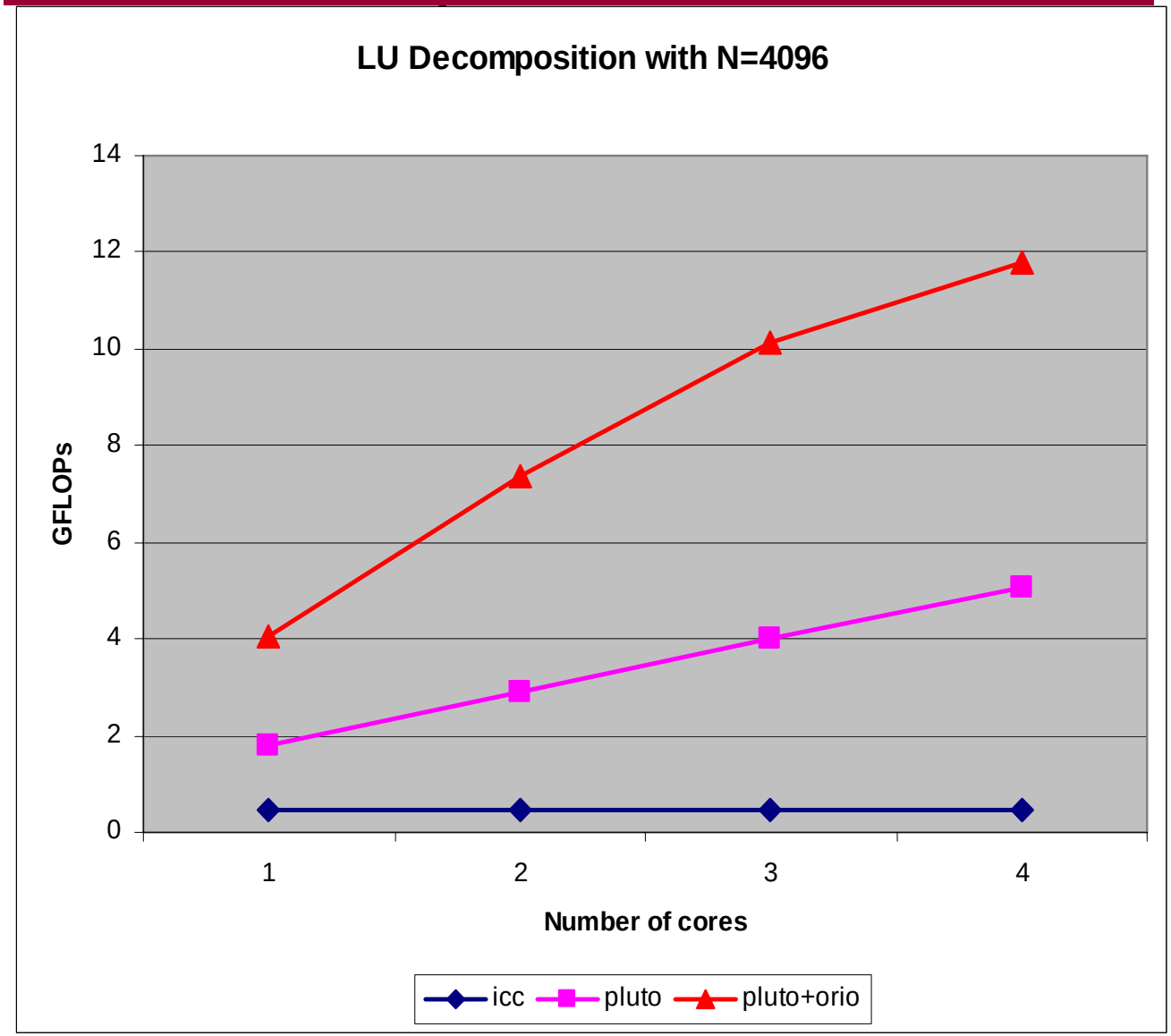
ADI Kernel: Multi-core



2-D FDTD: Multi-core

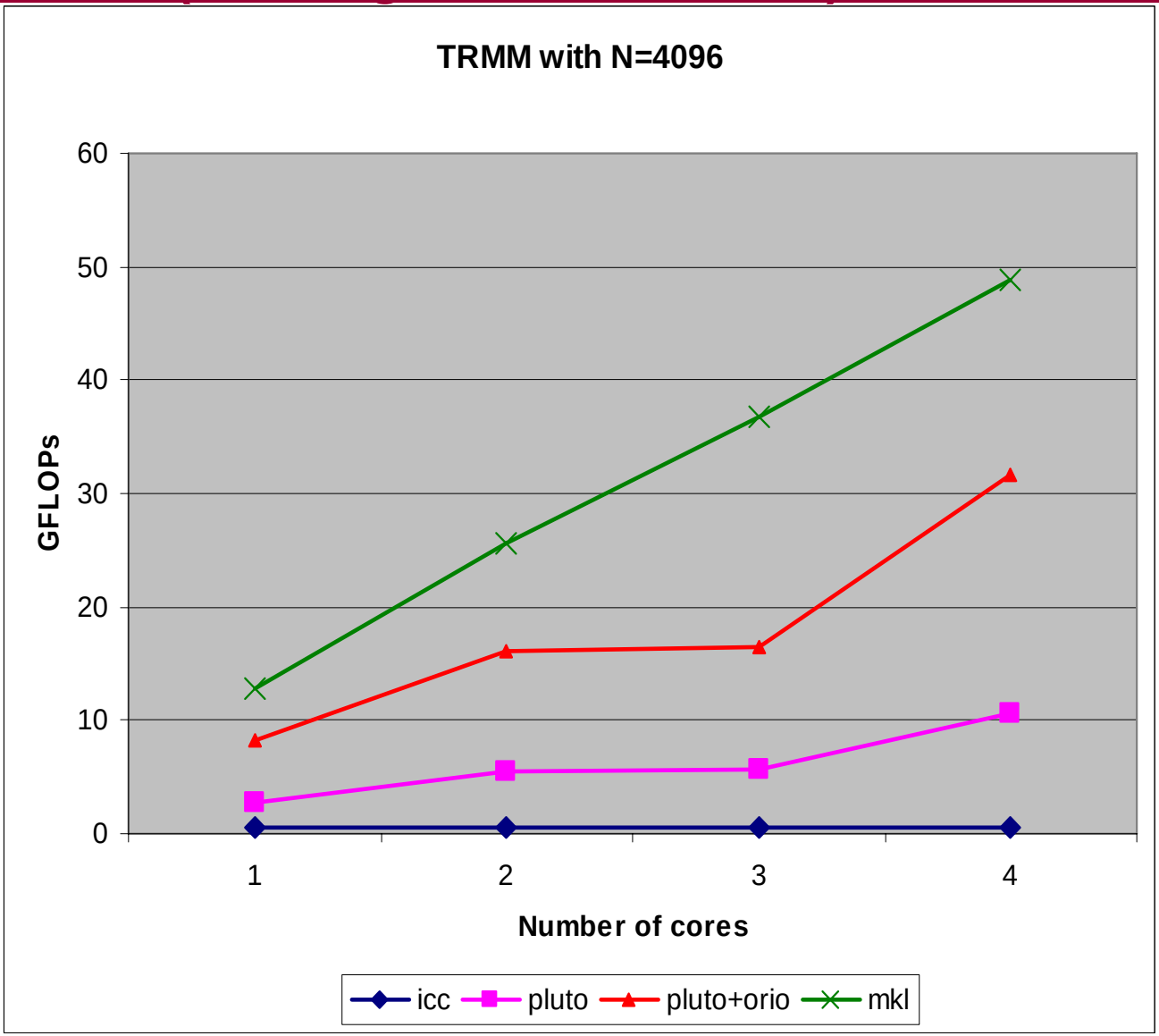


LU Decomposition: Multi-core



3-D Gauss Seidel: Multi-core

TRMM (Triangular MatMult): Multi-Core



Tensors Codes: BLAS + Index Permutations

- Highly-tuned GEMM routines in the BLAS library can be used since a tensor contraction is essentially a generalized matrix multiplication.

- GEMM requires a two-dimensional view of the input matrices:
  - Summation and non-summation indices should be grouped into two contiguous sets.
  - Index permutation is needed to reshape the arrays.

- Goal: Minimize the execution time of the generated code

Possible Approach

- Key aspects of this approach
  - Optimize a sequence of calls using information about the performance of these routines.
  - Provide portable performance across architectures.
- Two types of constituent operations:
  - Generalized Matrix Multiplication (GEMM)
  - Index Permutation
- Challenge: Useful, combinable empirical performance-model of constituent operations.
  - Optimize index permutation + choice of GEMM
  - Sequence of tensor contractions
  - Exploiting parallelism

Example: BLAS + index permutations

A contraction example:

$$E(i,\ j,c) = \sum_{a,b} [A(a,b,c) \times B(a,i) \times C(b,\ j)]$$

All indices range over N, an operation-minimal evaluation sequence is:

$$T1(i,b,c) = \sum_{a} [A(a,b,c) \times B(a,i)]$$

$$E(i,\ j,c) = \sum_{b} [T1(i,b,c) \times C(b,\ j)]$$

Example: BLAS + index permutations

Many ways of generating code, two of them are:

1:

Reshape A: (a,b,c)→(c,b,a)

GEMM: B(a,i) x A(cb,a) → T1(i,cb); with (t,t)

GEMM: C(b,j) x T1(ic,b) → E(j,ic); with (t,t)

Reshape E: (j,i,c)→(i,j,c)

2:

GEMM: A(a,bc)xB(a,i)→T1(bc,i); with (t,n)

GEMM: T1(b,ci)xC(b,j)→E(ci,j); with (t,n)

Reshape E: (c,i,j)→(i,j,c)

Neither one is better than the other for all the array sizes!

Experimental Evaluation

Processor	OS	Compiler	Inter-connect	Comm. Lib	BLAS Lib
Dual Itanium 2 900 MHz	linux 2.4.18	efc 7.1	Myrinet 2000	GA/ARMCI	MKL 5.1

$$T1(a,q,r,s) = \sum_{p} C4(p,a) * A(p,q,r,s)$$
$$T2(a,b,r,s) = \sum_{q} C3(q,b) * T1(a,q,r,s)$$
$$T3(a,b,c,s) = \sum_{r} C2(r,c) * T2(a,b,r,s)$$
$$B(a,b,c,d) = \sum_{s} C1(s,d) * T3(a,b,c,s)$$

- Atomic-Orbital to Molecular-Orbital Integral transform: very important transformation in quantum chemistry codes
- Tensors (double precision elements):
  - Sequential experiments: $N_p = N_q = N_r = N_s = N_a = N_b = N_c = N_d = 64$
  - Parallel experiments: $N_p = N_q = N_r = N_s = N_a = N_b = N_c = N_d = 96$

Experimental Results

- Sequential results: the improvement is 20%

Unoptimized (sec.)				Optimized (sec.)			
GEMM	Index Permutation	Exec. Time	GFLOPS	GEMM	Index Permutation	Exec. Time	GFLOPS
10.06	2.58	12.64	2.07	10.58	0.0	10.58	2.48

- Parallel results on 4 processors: the improvement is 78%

Unoptimized (sec.)				Optimized (sec.)			
GEMM	Index Permutation	Exec. Time	GFLOPS	GEMM	Index Permutation	Exec. Time	GFLOPS
12.23	7.74	19.97	3.27	7.57	3.64	11.21	5.83

More Challenges

- Tensors are not always dense!
- Here are some challenges
  - Exploiting symmetry
  - Exploiting sparsity
  - Exploiting block-sparsity (RINO: Regular Inner Nonregular Outer computations)
- Appears to require combination of domain-specific information, architecture-aware optimizations, and machine-specific optimizations

References (1)

- U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A Practical and Automatic Polyhedral Program Optimization System," *Proc. ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation (PLDI 08),* Tucson, June 2008.
- U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, "Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model," in *Proc. CC 2008 - International Conference on Compiler Construction,* Budapest, Hungary, March-April 2008.
- A. Auer, G. Baumgartner, D. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Krishnamoorthy, S. Krishnan, C. Lam, Q. Lu, M. Nooijen, R. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov, "Automatic Code Generation for Many-Body Electronic Structure Methods: The Tensor Contraction Engine," *Molecular Physics,* vol. 104, no. 2, pp. 211--228, January 2006.
- G. Baumgartner, A. Auer, D. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Hirata, S. Krishnamoorthy, S. Krishnan, C. Lam, Q. Lu, M. Nooijen, R. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov, "Synthesis of High-Performance Parallel Programs for a Class of ab initio Quantum Chemistry Models," *Proceedings of the IEEE,* vol. 93, no. 2, pp. 276-292, February 2005.

References (2)

- A. Hartono, A. Sibiryakov, M. Nooijen, G. Baumgartner, D.E. Bernholdt, S. Hirata, C. Lam, R. Pitzer, J. Ramanujam, and P. Sadayappan, "Automated Operation Minimization of Tensor Contraction Expressions in Electronic Structure Calculations," in *Proc. International Conference on Computational Science 2005 (ICCS 2005)*, Atlanta, GA, May 2005.
- Q. Lu, X. Gao, S. Krishnamoorthy, G. Baumgartner, J. Ramanujam, and P. Sadayappan, "Empirical Performance-Model Driven Data Layout Optimization," *Languages and Compilers for Parallel Computing,* (R. Eigenmann et al. Eds.), Lecture Notes in Computer Science, Springer-Verlag, 2005.
- A. Bibireata, S. Krishnan, G. Baumgartner, D. Cociorva, C. Lam, P. Sadayappan, J. Ramanujam, D. Bernholdt, and V. Choppella, "Memory-Constrained Data Locality Optimization for Tensor Contractions," in *Languages and Compilers for Parallel Computing,* (L. Rauchwerger et al. Eds.), Lecture Notes in Computer Science, Vol. 2958, pp. 93-108, Springer-Verlag, 2004.
- Chi-Chung Lam, P. Sadayappan, Rephael Wenger: Optimal Reordering and Mapping of a Class of Nested-Loops for Parallel Execution," in *Languages and Compilers for Parallel Computing,* (D. Sehr et al. Eds.), Lecture Notes in Computer Science, Vol. 1239, pp. 315-329, Springer-Verlag, 1997.

Thank You