A polyhedral loop transformation framework for parallelization and tuning

Ohio State University

Uday Bondhugula, Muthu Baskaran, Albert Hartono, Sriram Krishnamoorthy, P. Sadayappan

Argonne National Laboratory

Boyana Norris

Louisiana State University

J. Ramanujam

Supported by the National Science Foundation

Questions/Propositions

- Disagree
  - "Parameter tuning" is the wrong focus for our area, and will lead only to incremental improvements
  - Simple performance models (e.g., a cache-oblivious model) will be the right models in the future and will obviate the need for empirical search.
  - The focus on specialized tuning systems is too narrow, and so only compilers, which apply most broadly, are the most sensible investment [One does not preclude the other]
- What other technologies should we investigate to find application-specific, platform-specific improvement?
  - High-level languages and libraries may be the only effective way to achieve high performance on peta/exascale systems

Automatic Parallelization of Sequential Code

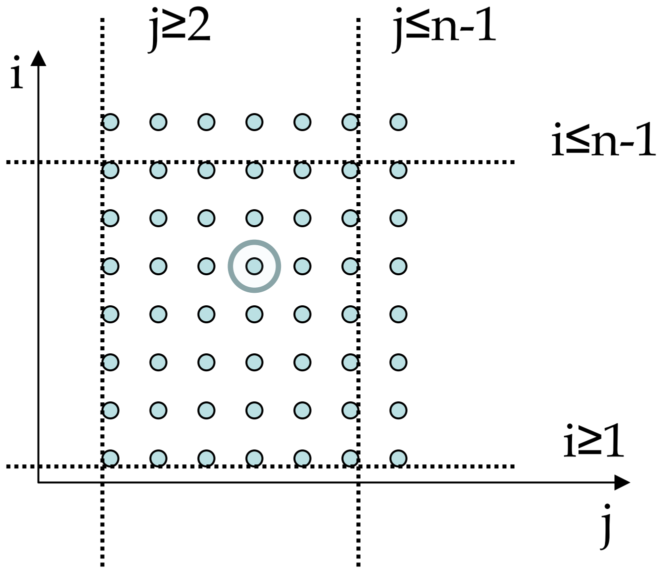
- Automatic parallelization has been a long-sought goal
  - Large body of compiler optimization research
  - Heightened interest now with ubiquity of multi-core processors
- Several vendor compilers offer an automatic parallelization feature for SMP/multi-core systems
  - Limited use in practice; users 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

Polyhedral Compiler Framework

- Powerful abstraction for data dependences and program transformation
- Unified treatment of many loop transforms
- Effective handling of imperfectly nested loops
- Natural handling of parametric loop bounds
- Proposed in the early 90's; initially considered impractical for production optimizing compilers
- Recent advances have addressed many issues of compilation overhead as well as quality of generated code

Polyhedral Model

```
for (i=1; i<n; i++)
  for (j=2; j<n; j++)
      S1: a[i][j] = a[j][i] + a[i][j-1];
```



$$\vec{x}_{S1} = \begin{pmatrix} i \\ j \end{pmatrix}$$

$$I_{S1} = \begin{pmatrix} 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & -1 \\ 0 & 1 & 0 & -2 \\ 0 & -1 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} \vec{x}_{S1} \\ n \\ 1 \end{pmatrix} \geq \vec{0}$$

Stmt instances ⇔integer points in polyhedra ⇔ systems of linear inequalities

Polyhedral Model - 2

```
for (i=1; i<n; i++)
  for (j=2; j<n; j++)
    S1: a[i][j] = a[j][i] + a[i][j-1] ;
```

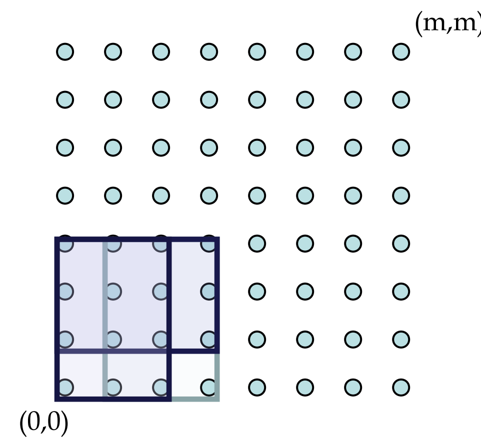
$$\mathcal{I}_{S1} = \begin{pmatrix} 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & -1 \\ 0 & 1 & 0 & -2 \\ 0 & -1 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} \vec{x}_{S1} \\ n \\ 1 \end{pmatrix} \geq \vec{0}$$

$$F_{1a}(\vec{x}_{S1}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \vec{x}_{S1} \\ n \\ 1 \end{pmatrix}$$

$$\mathcal{DS}_{1a} = F_{1a}\,\mathcal{I}_{S1}$$

$$F_{2a}(\vec{x}_{S1}) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \vec{x}_{S1} \\ n \\ 1 \end{pmatrix}$$

$$F_{3a}(\vec{x}_{S1}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \vec{x}_{S1} \\ n \\ 1 \end{pmatrix}$$



Footprint of Array Reference ⇔ integer points in data space polyhedra

Previous Related Work

- Tiling has been widely studied: Schreiber ’90, Wolf-Lam ’91, Ramanujam et al. ’92, Boulet et al. ’94, Darte et al. ’97, Xue ’97, Hogstedt et al. ’99, Wonnacott et al. ’00, Song-Li ’99, Hodzic ’02, Andonov-Rajopadhye ’03, Yi et al. ’04, Renganarayana et al. ’04, . . .

- Polyhedral loop transformation and code generation: Feautrier 1991, Kelly-Pugh ’95,’98, Lim-Lam ’97,’99,’01, Quillere et al. ’00, Ahmed-Pingali ’00, Griebl ’04, Bastoul ’04, Cohen et al. ’05, Girbal et al. ’06, Pouchet et al. ’07, ’08

- But no practical and effective approach to tiling of general (affine) imperfectly nested loops for parallelism and locality

Polyhedral Transformation and Tiling

- Tiling is a key loop transformation for efficient coarse-grained parallel execution, and for data locality optimization

```
for i=1,N
  for j = 1,N
     { S(i,j) }
```

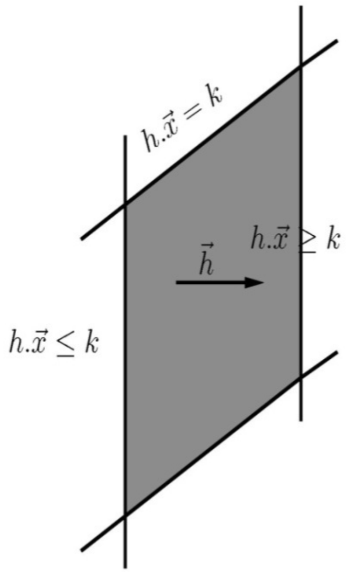
→

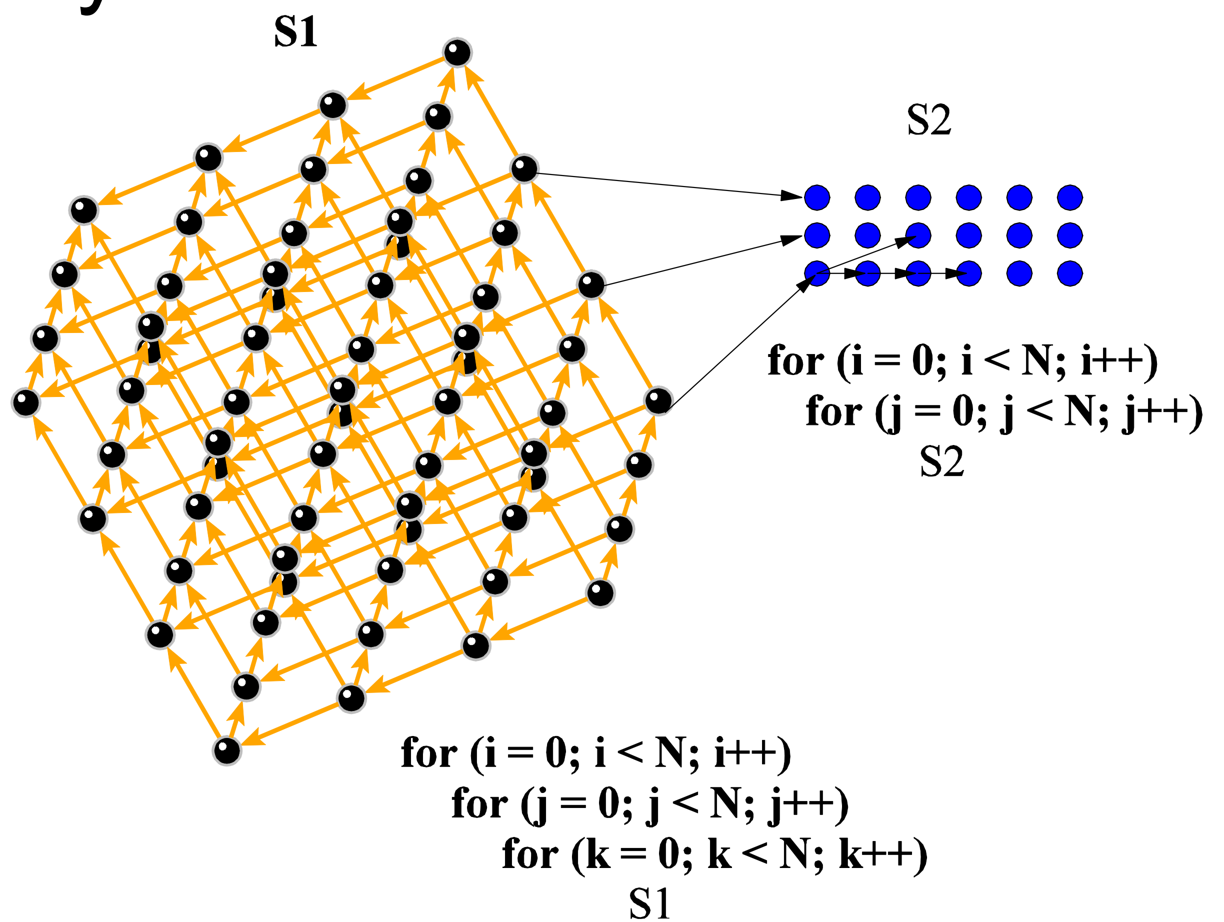
```
for it=1,N,T
  for jt = 1,N,T
    for ii = it,min(it+T-1,N)
      for jj = jt,min(jt+T-1,N)
        { S(ii,jj) }
```

- Previously, tiling was treated as a post-processing step on permutable loop-nests in polyhedral transformation frameworks: no practically effective tiling algorithm

- Our recent work (CC ’08, PLDI ’08) has developed a model-driven approach to automatically tile imperfectly nested (affine) loops for parallelism and data locality

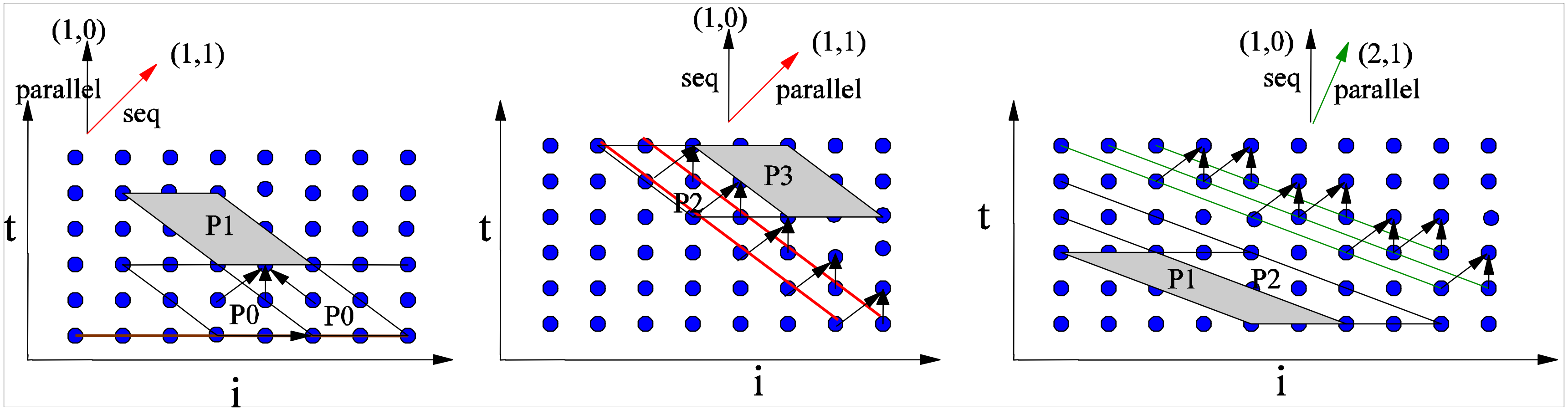
Polyhedral Model





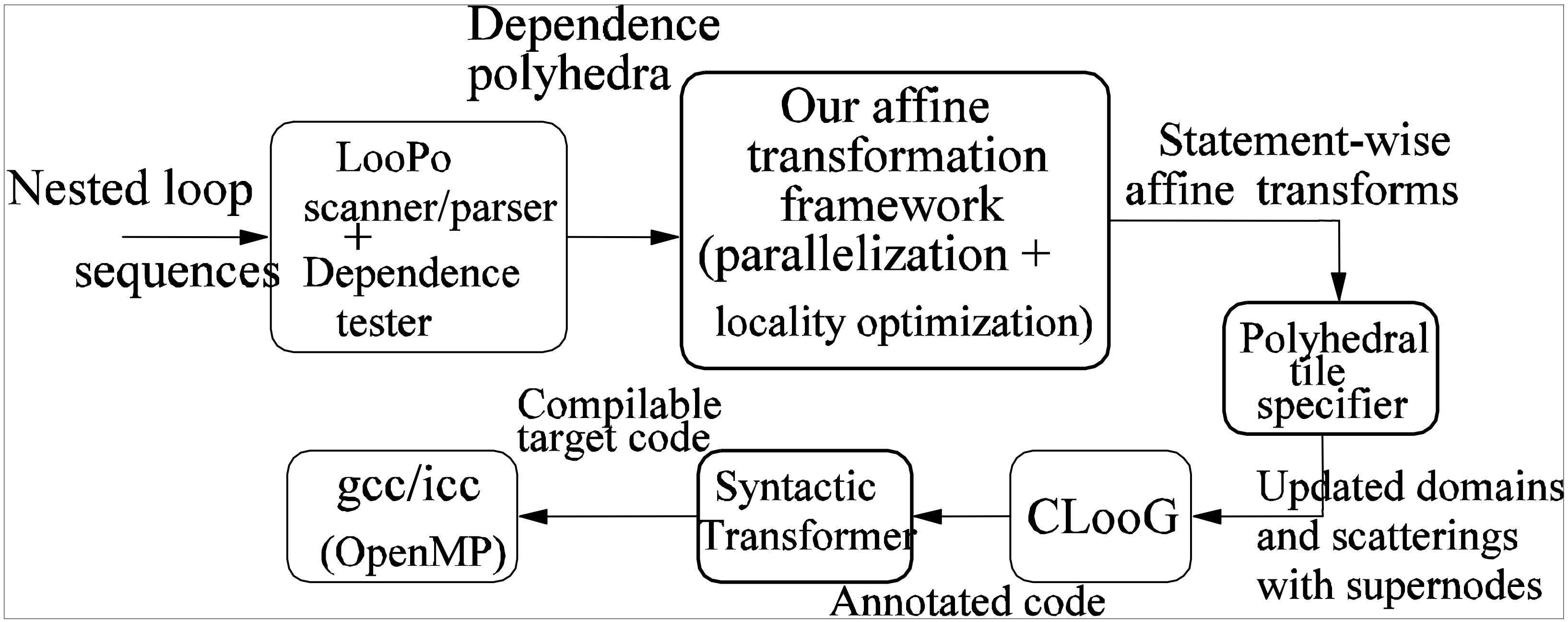
- Each stmt in arbitrarily imperfectly nested loop viewed as a polyhedron
- Dependences modeled as polyhedra in higher-dim space
- Stmt. instances mapped to multi-dim time via affine scheduling function
- Tiling hyperplanes are equiv. to scheduling functions with some properties
- Use parametric ILP machinery to optimize hyperplanes for //sm & locality

Communication Volume & Reuse Distance



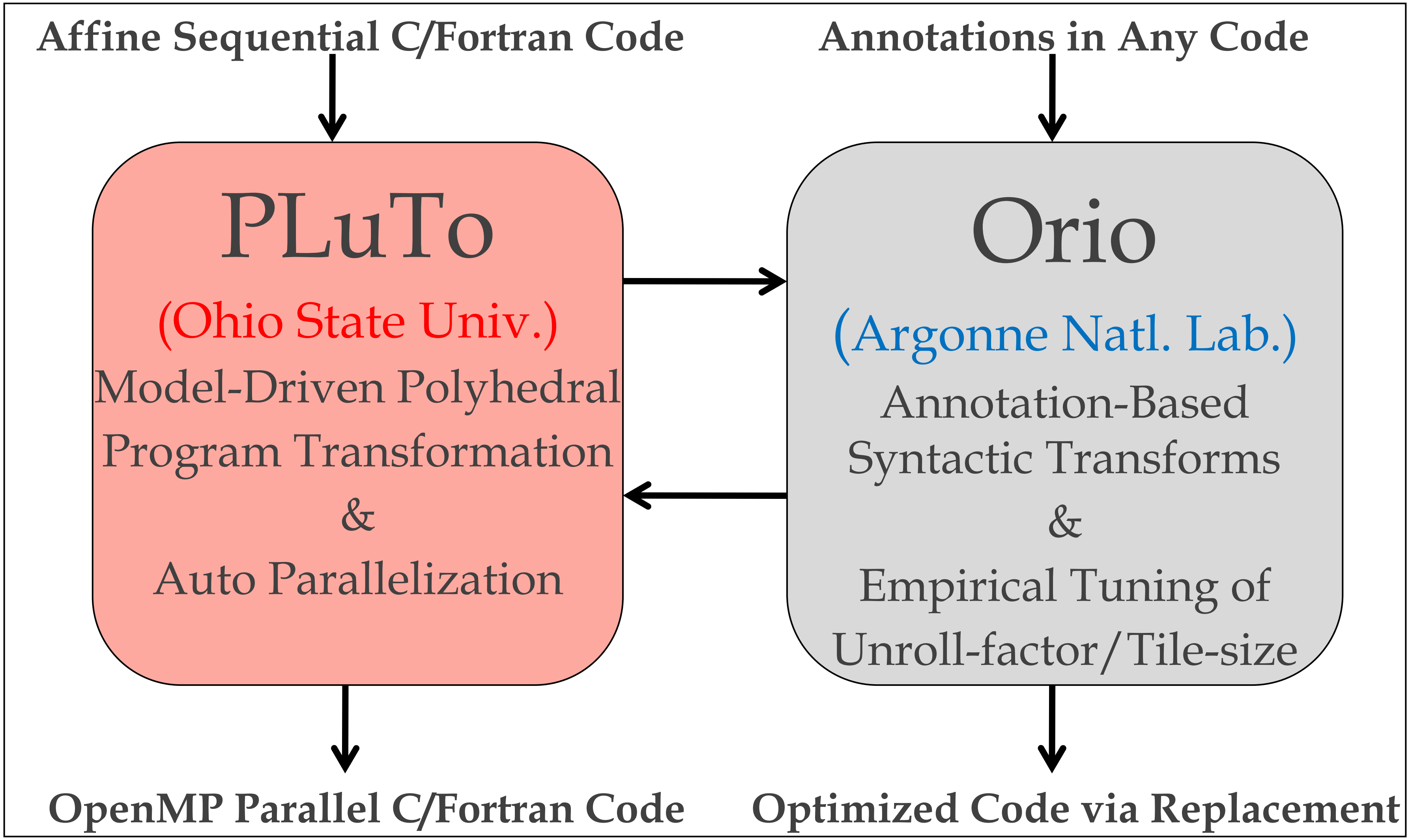
- $\phi(\mathbf{i}) - \phi(\mathbf{i}')$ represents the component of a dependence along the hyperplane
  - Communication volume (per unit area) at processor tile boundaries
  - Cache misses at local tile edges

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



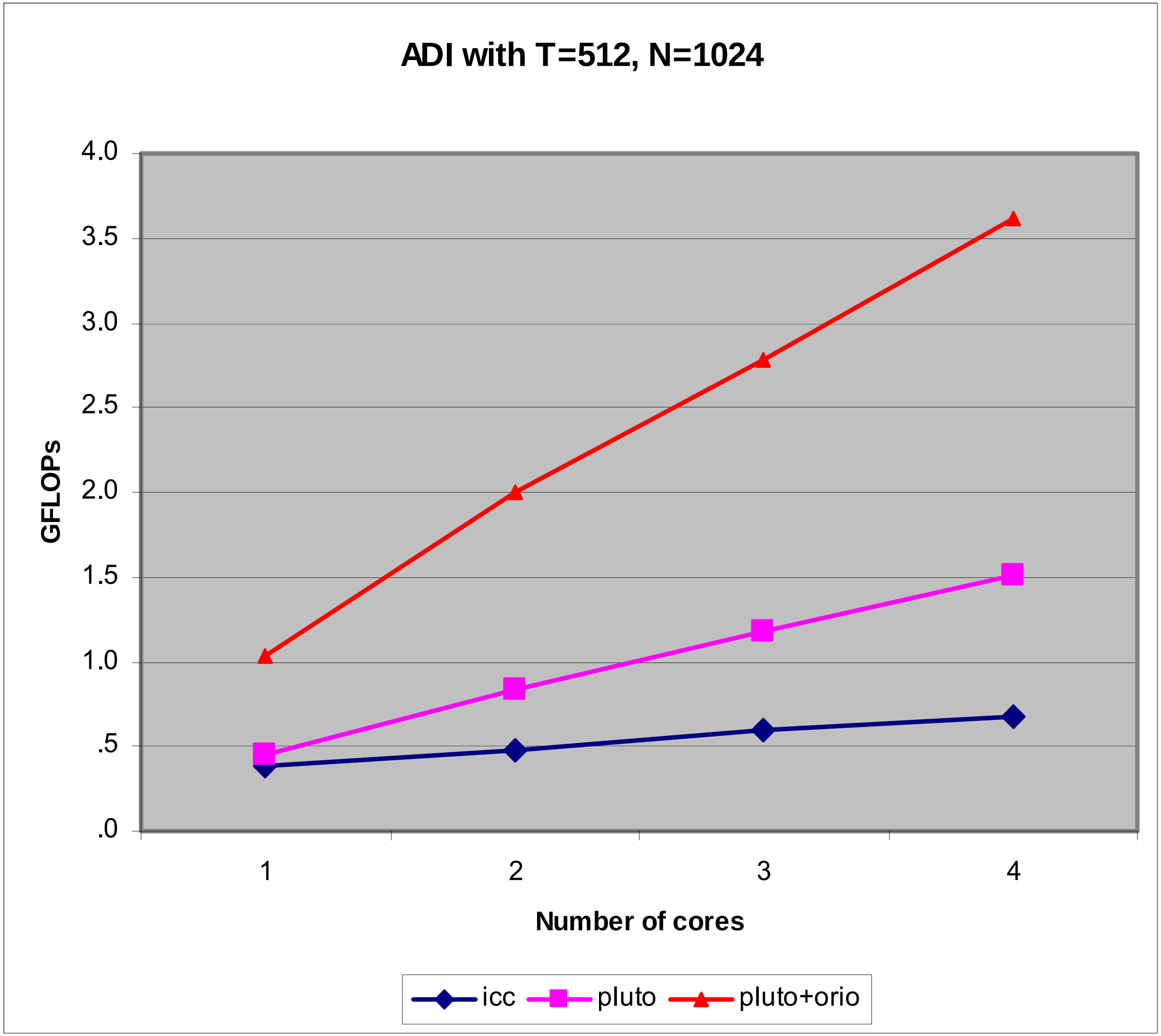
http://sourceforge.net/projects/pluto-compiler

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

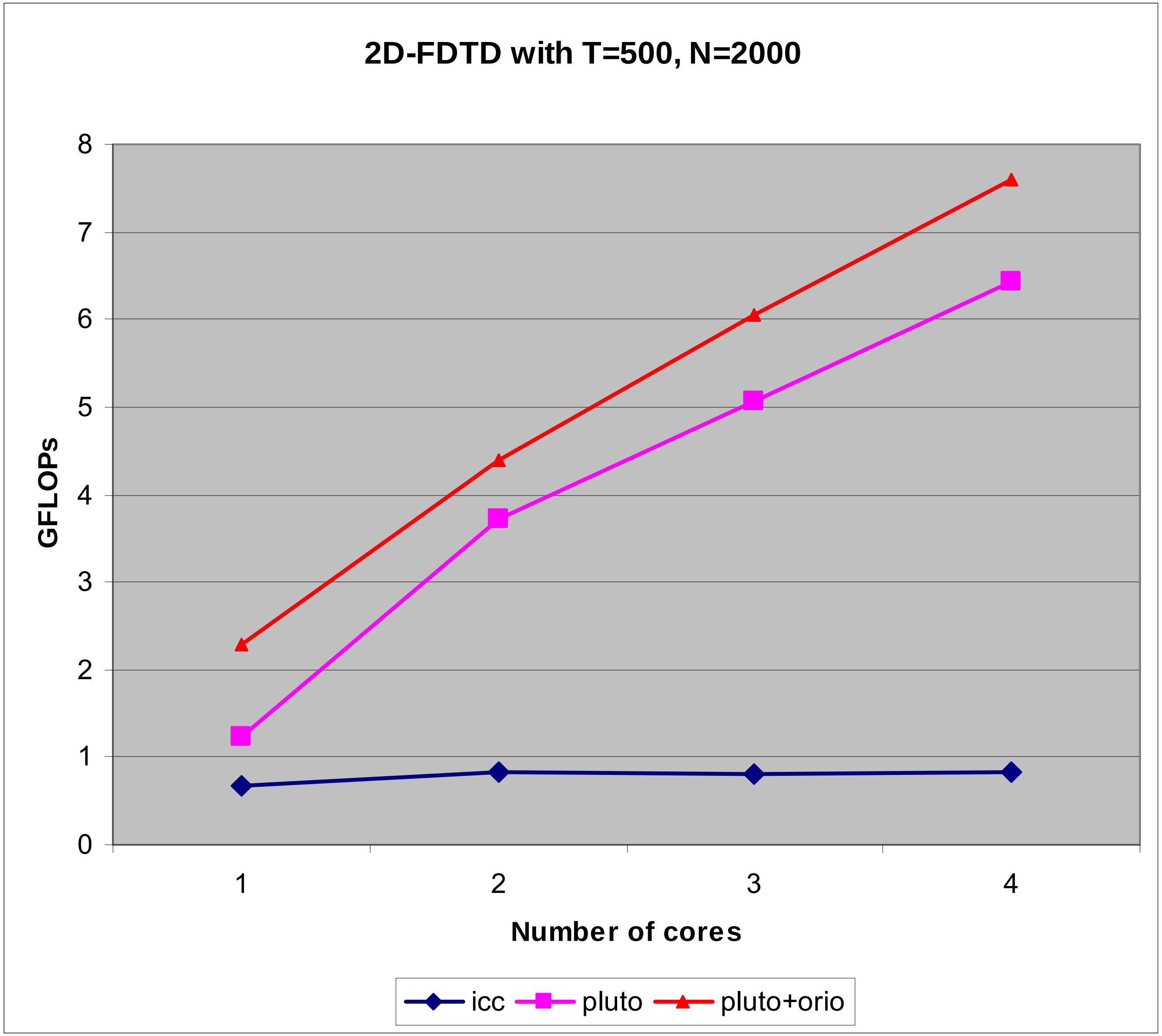
Experimental Results

- 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

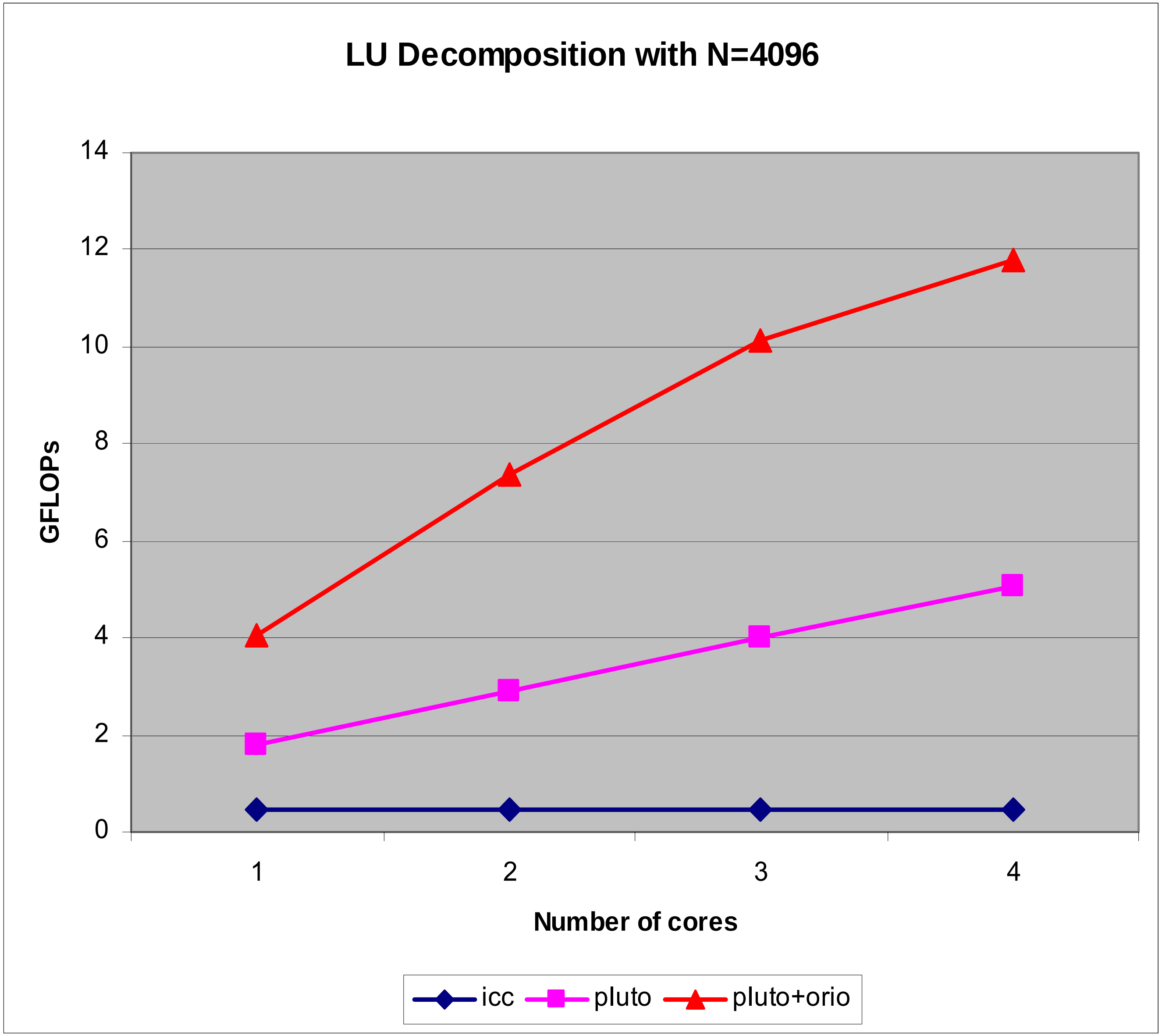
ADI Kernel: Multi-core



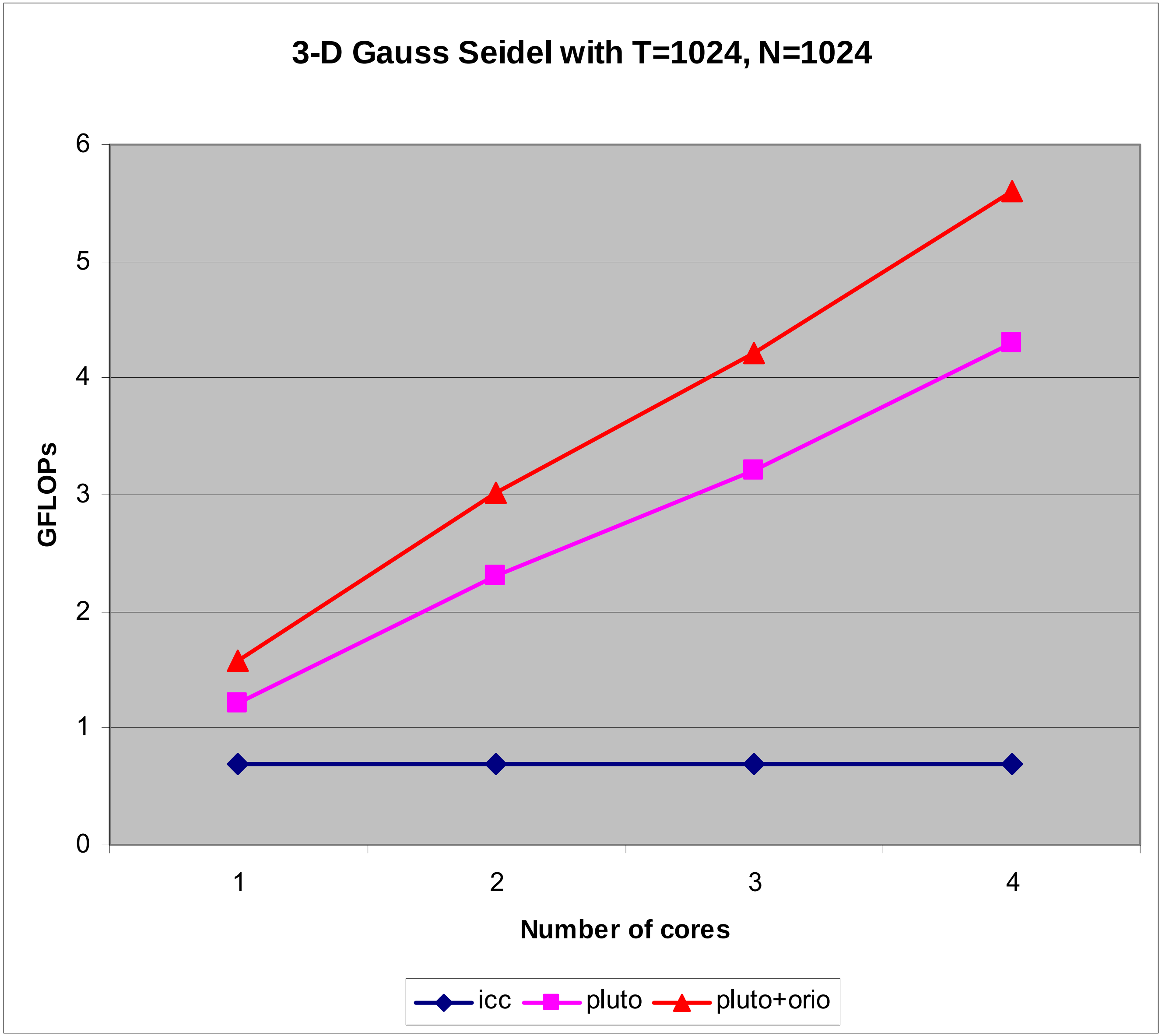
2-D FDTD: Multi-core



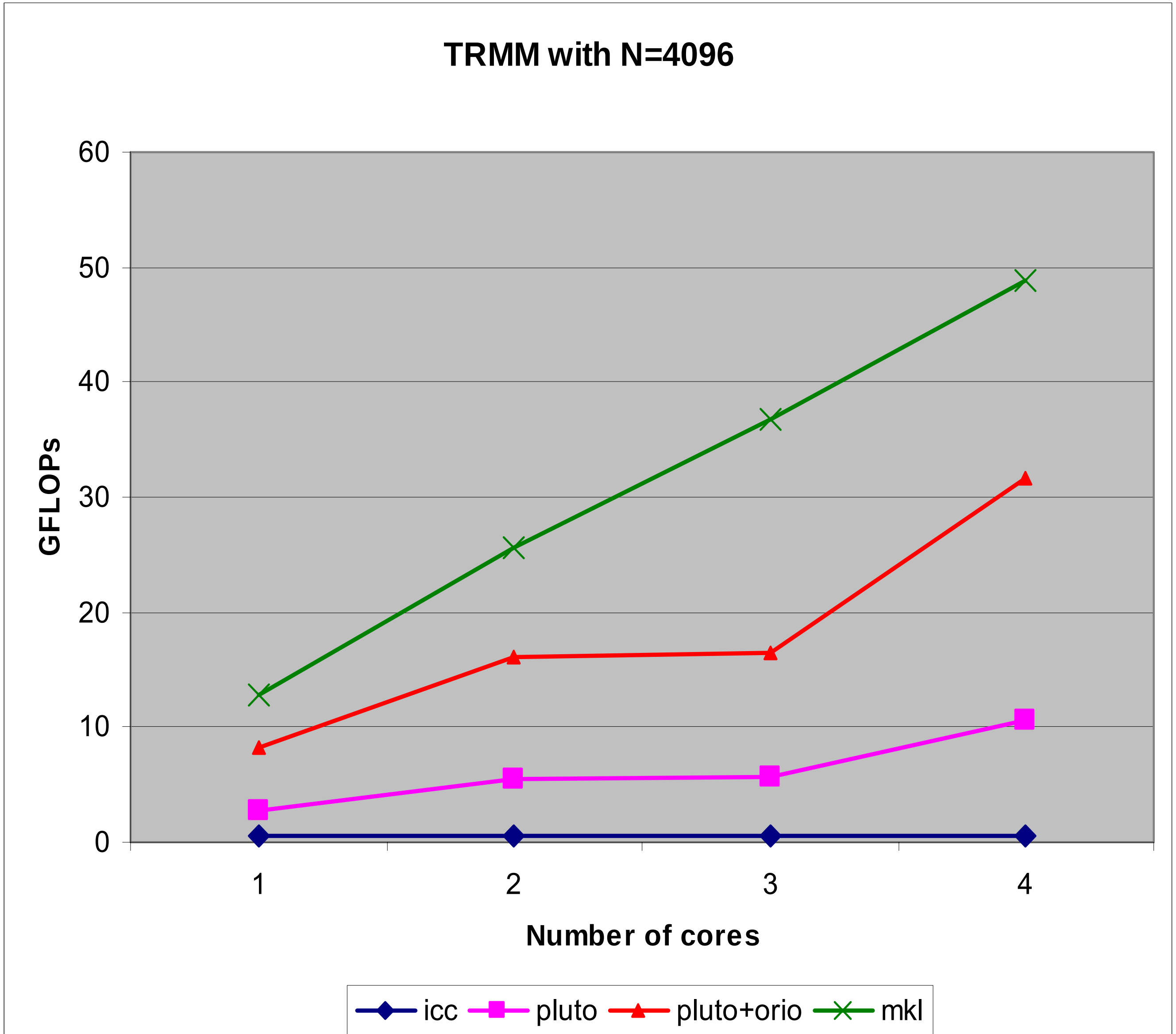
LU Decomposition: Multi-core



3-D Gauss Seidel: Multi-core



TRMM (Triangular MatMult): Multi-Core



How prevalent are affine codes?

- Innermost core computations in many codes
  - Dense linear algebra
  - Image and signal processing
  - Computational Electromagnetics (FDTD)
  - Explicit PDE solvers (e.g. SWIM, SWEEP3D)
  - Integral transforms in quantum chemistry (AO-to-MO)
- May increase in the future (esp. scientific apps)
  - Codes with direct data access significantly better than indirect-data access: power & performance
  - Structured-sparse (block sparse) is better than arbitrary sparse (e.g. OSKI)
  - RINO (Regular-Inner-Nonregular-Outer) algorithms should be attractive for many-core processors

Summary

- Polyhedral compiler optimization framework
  - New approach to effective tiling for parallelism and data locality (CC-08, PLDI-08)
- Automatic parallelization tool for multi-cores
  - http://sourceforge.net/projects/pluto-compiler/
- Promising basis for model-driven empirical tuning
  - Has been coupled with Orio annotation-based syntactic transformation & tuning tool
- Many extensions under exploration
    - Generate CUDA code for GPGPUs (next talk by Ram)
    - Dynamic scheduling with self-extracted inter-tile dependences
    - Iterative opt. with dynamic trace analysis, user feedback

Thank you