

Construction and Performance Characterization of Parallel Interior Point Solver on 4-way Intel Itanium 2

P. Koka

Department of ECE, University of Wisconsin Madison

T. Suh

Department of ECE, Georgia Institute of Technology

M. Smelyanskiy R. Grzeszczuk C. Dulong
Systems Technology Labs - Intel



Linear Programming

- Linear programming problem

$$\min_{\mathbf{x}} f(\mathbf{x}) = c_1 * x_1 + \dots + c_n * x_n = \mathbf{c}^T \mathbf{x} - \text{objective function}$$

subject to $\mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$ – constraints

where \mathbf{A} is an $m \times n$ matrix, and $m \leq n$, i.e., both constraints and objective function are linear

- Applications of linear programming

Area

Scheduling

Medicine &
Biology

Telecommu-
nications

Finance

Examples

Crew Scheduling
Production and
Inventory
Compiler Scheduling

Radiation Therapy
Protein Folding,
Drug Design

Call Routing
Network Design
Internet Traffic

Portfolio
Optimization
Option Pricing

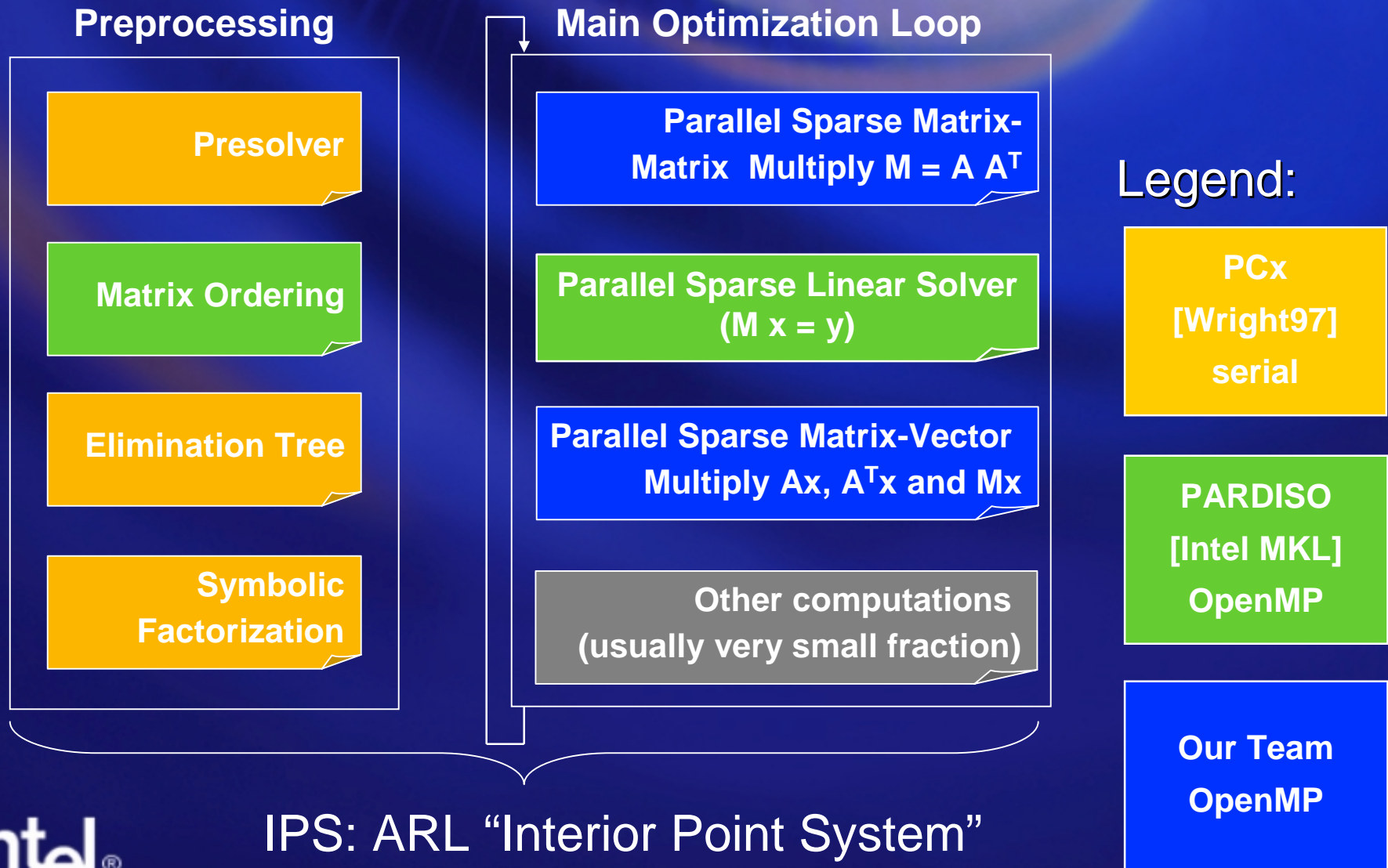
Important Characteristics of LPs

- LPs arising in many practical application today have certain structure that enables parallelism
- Very large problems
 - > 10M of variables, > 1M constraints
- Very sparse problems
 - A small number of variables in each column is non-zero
 - Constraint matrix A is very sparse

Linear Programming Solvers

- **Simplex method [Danzig47, Lemke54]**
 - Started modern era of optimization
 - bad “worst-case” behavior (but rare in practice)
 - difficult to parallelize
- **Interior point method (IPM) [Karmarkar84]**
 - responsible for most of recent progress in optimization not only of linear programming
 - excellent theoretical properties
 - good practical performance (converges in 50-70 iterations)
 - high potential scalability on many processors

IPM Parallel Workload Building Blocks



Parallel Sparse Matrix-Matrix and Matrix-Vector Multiply

- Sparse matrices A and M are stored in compressed storage format
- Use OpenMP with dynamic scheduling to load balance
- Parallel Matrix-Matrix Multiply ($M = A A^T$)
 - $M_{ij} = A_i \cdot A_j$ - dot product of two columns i and j of A
 - several consecutive elements of M are assigned to the same thread
- Parallel Matrix-Vector Multiply ($y = A x$)
 - $y_i = A_i \cdot x$ - dot product of column i of A and vector x
 - matrix A is partitioned row-wise among threads

Sparse Linear Solver

- Solve linear system $M \mathbf{x} = \mathbf{b}$, where M is sparse positive definite
- Algorithm:
 1. Cholesky factorization: $M = LL^T$, L is lower triangular
 2. Solve $L \mathbf{y} = \mathbf{b}$ through forward substitution
 3. Solve $L^T \mathbf{x} = \mathbf{y}$ through backward substitution
- L can have much more non-zeros (fill-ins) than M
 - **Matrix ordering**: find permutation matrix P such that L factor of PMP^T have reduced fill-in
 - Ordering is performed once in preprocessing step

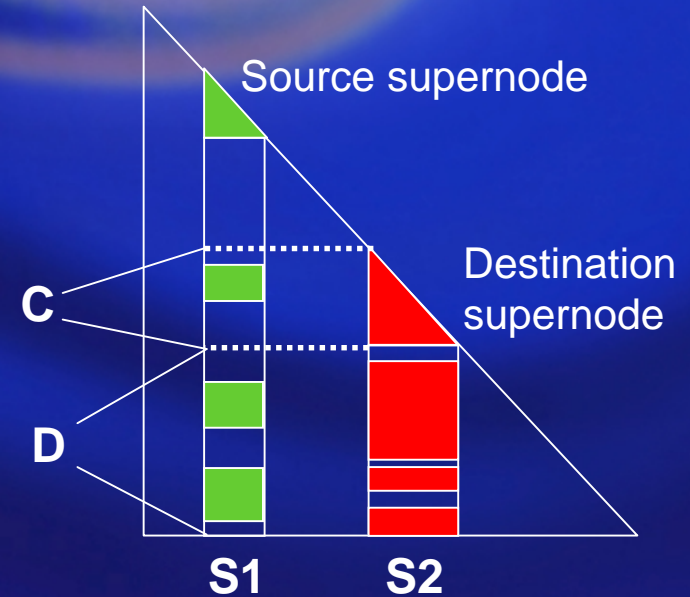
Parallel Sparse PARDISO Cholesky

Example of Supernodes

M =

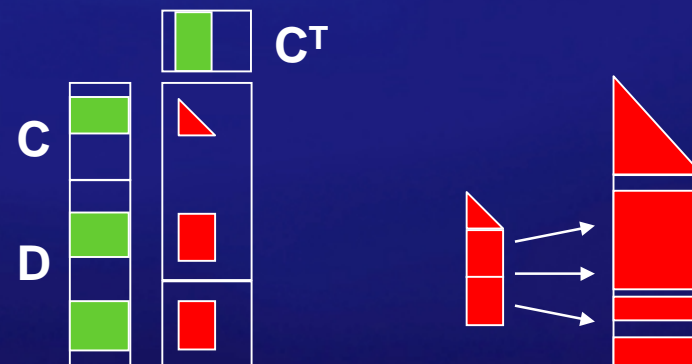
	1	2	3	4	5	6	7	8
1	1							
2	2	3						
3	5	1	4					
4			1	3				
5	1	4	2	2	7			
6			3	7	9	8		
7						4	5	
8	2	7	2	4	1	1	8	9
	S1	S2	S3					

Supernode-Supernode Update



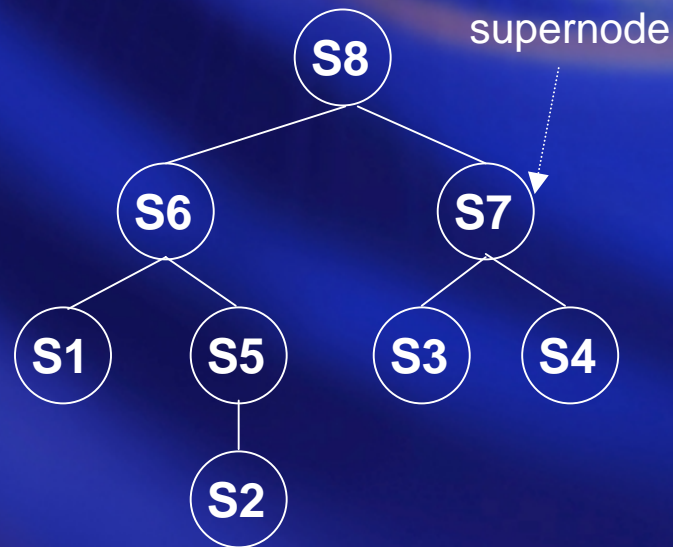
1. dense matrix-matrix multiply

2. scatter



Parallel Sparse PARDISO Cholesky

Elimination Tree



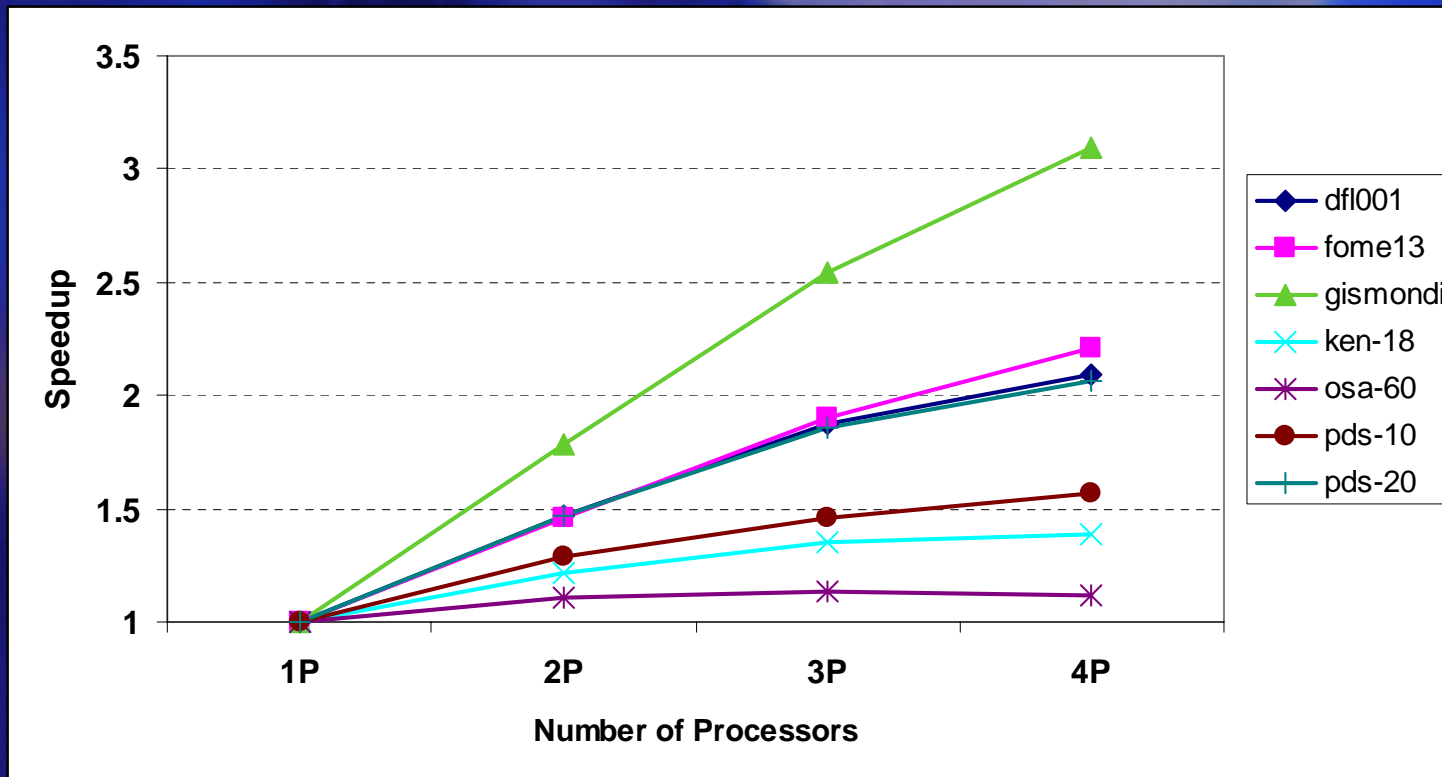
- A supernode is finished when its descendants are finished and **updated** the supernode
- Finished supernode generates updates to its ancestor supernodes
- Independent supernodes can be worked on in parallel

Performance Analysis of IPS

- 4-way Itanium system running Linux OS
 - 1Ghz, 16KB I/D L1, 256KB unified L2, 3MB unified L3 and 8GB memory
- Intel C & Fortran compilers with `-O2` and `-omp` flags
- Use VTune Start/Stop API system profiler
 - Warm up for 2 iterations and sample for the next 12 iterations
- Datasets

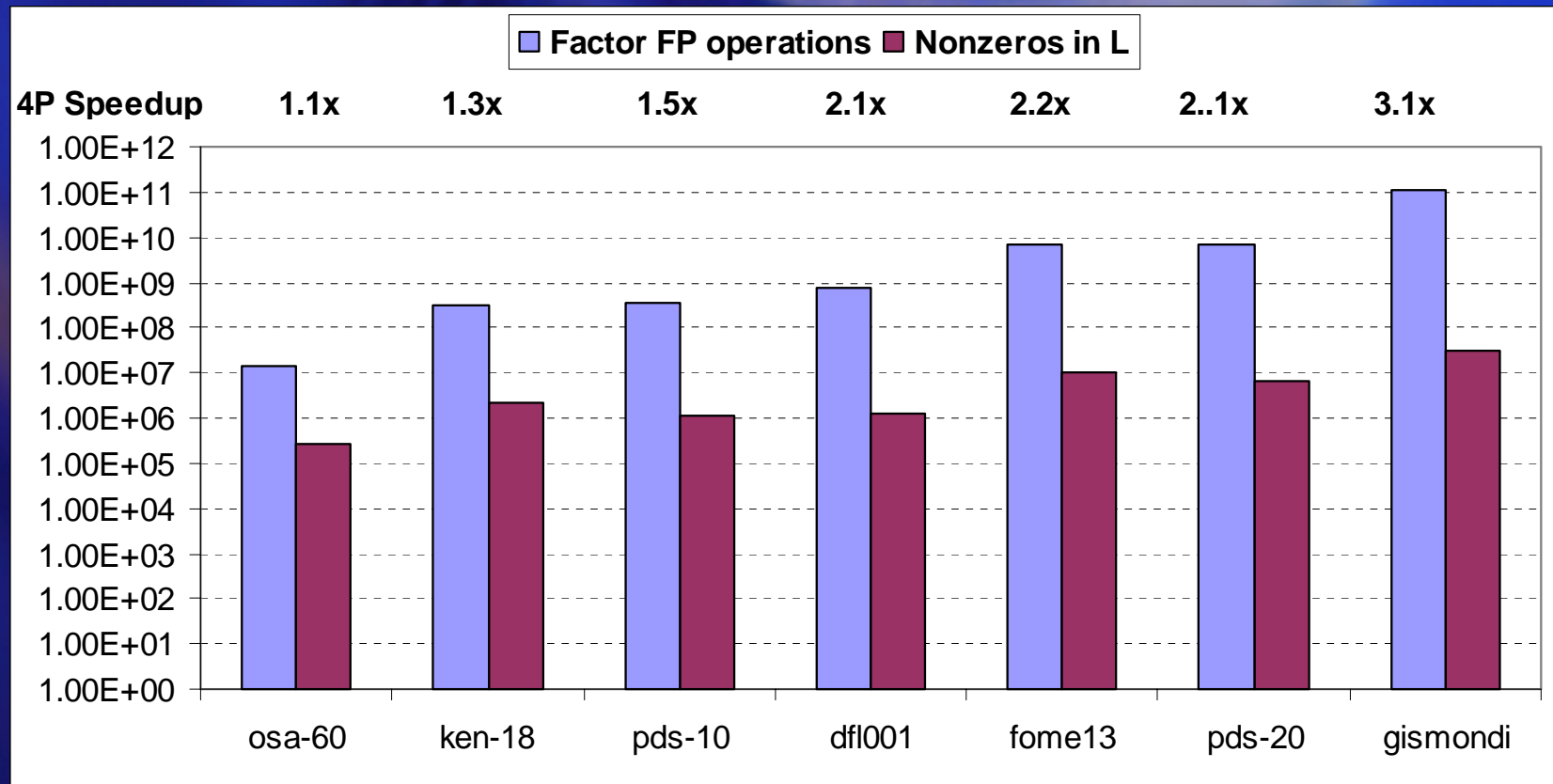
Problem	Rows	Columns	Nonzeros in A	% nonzeros
df1001	6071	12230	35632	4.80E-02
fome13	48568	97840	285056	6.00E-03
gismondi	18262	23211	136324	3.22E-02
ken-18	105127	154699	358171	2.20E-03
osa-60	10280	232966	1397793	5.84E-02
pds-10	16558	48763	106436	1.32E-02
pds-20	33874	105728	230200	6.43E-03

Speedup on 4-way IPF



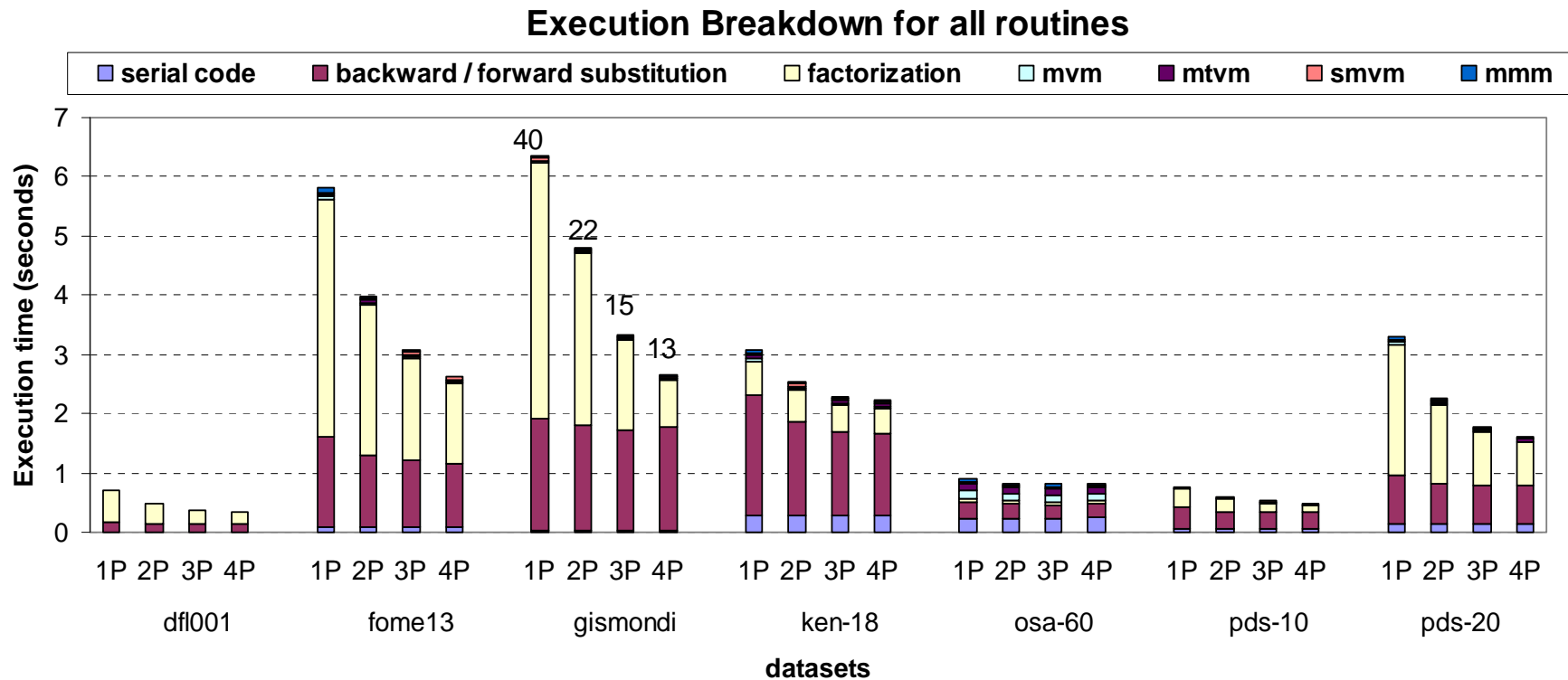
- Large variations in scalability

Dataset Characteristics vs. Scalability



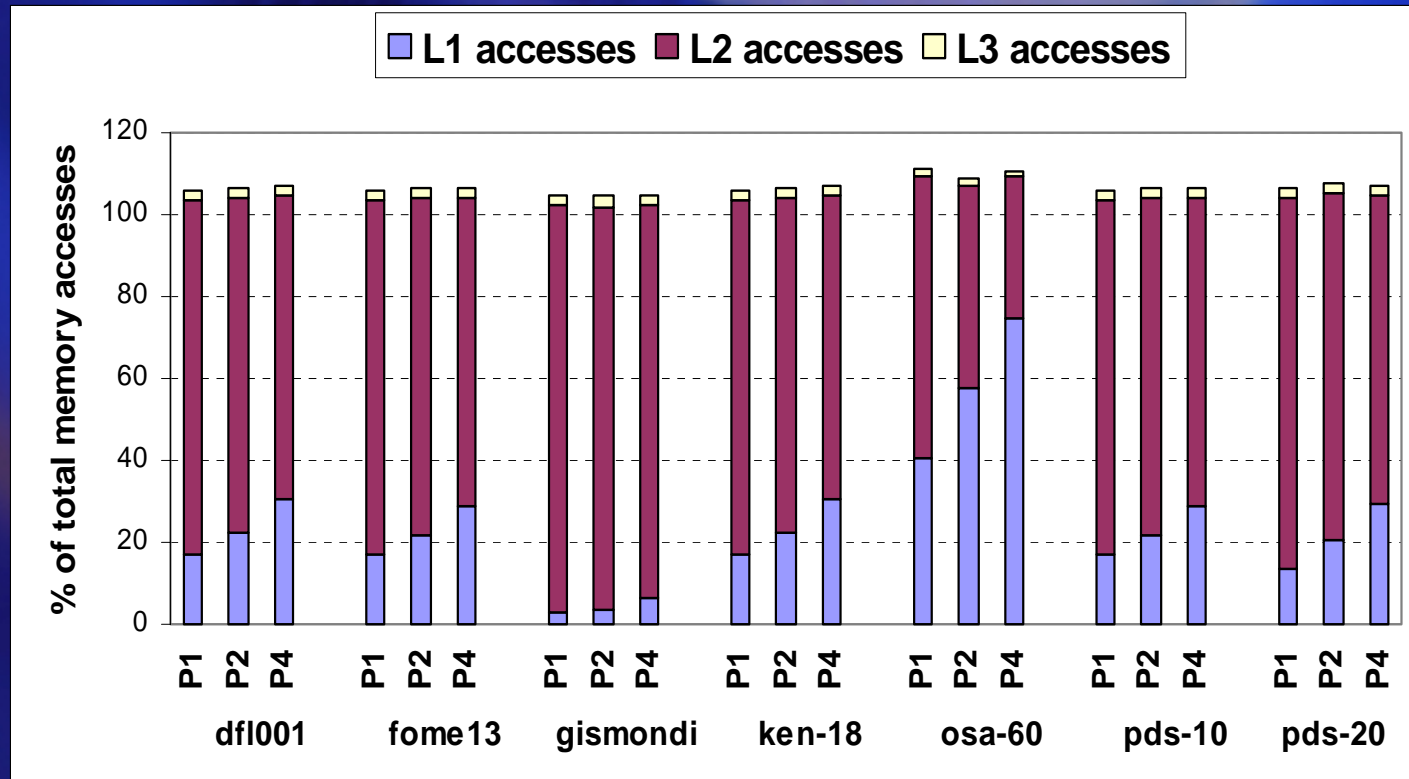
- Scalability directly correlates with number of FLOPS required to factorize the matrix
- FLOPS required correlates with number of non-zeros in L

Execution Time Breakdown



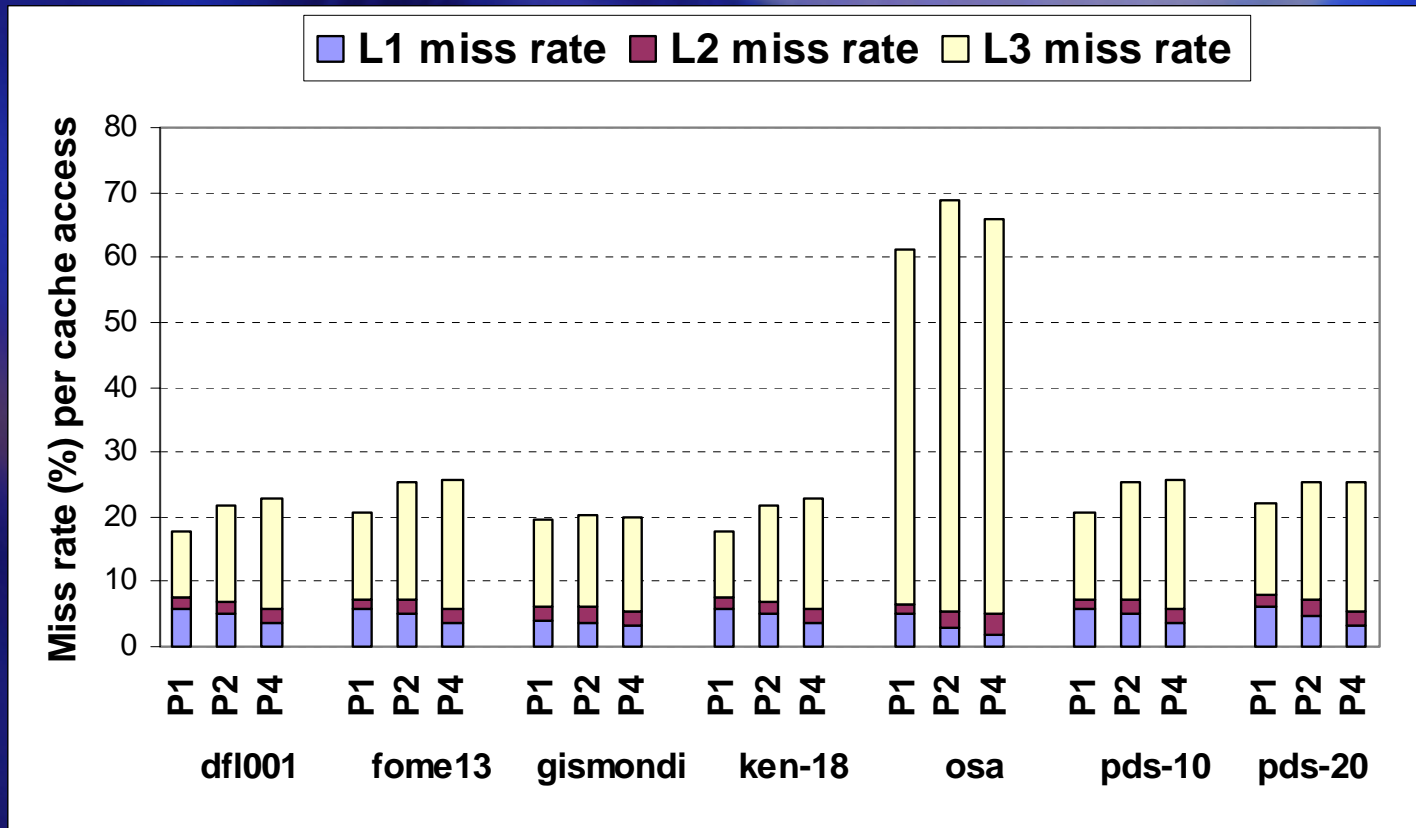
- At least 90% of the execution time is spent in parallel regions
- Factorization scales well
- Solver takes substantial part of execution time

Cache Behavior: L1, L2, L3 Accesses



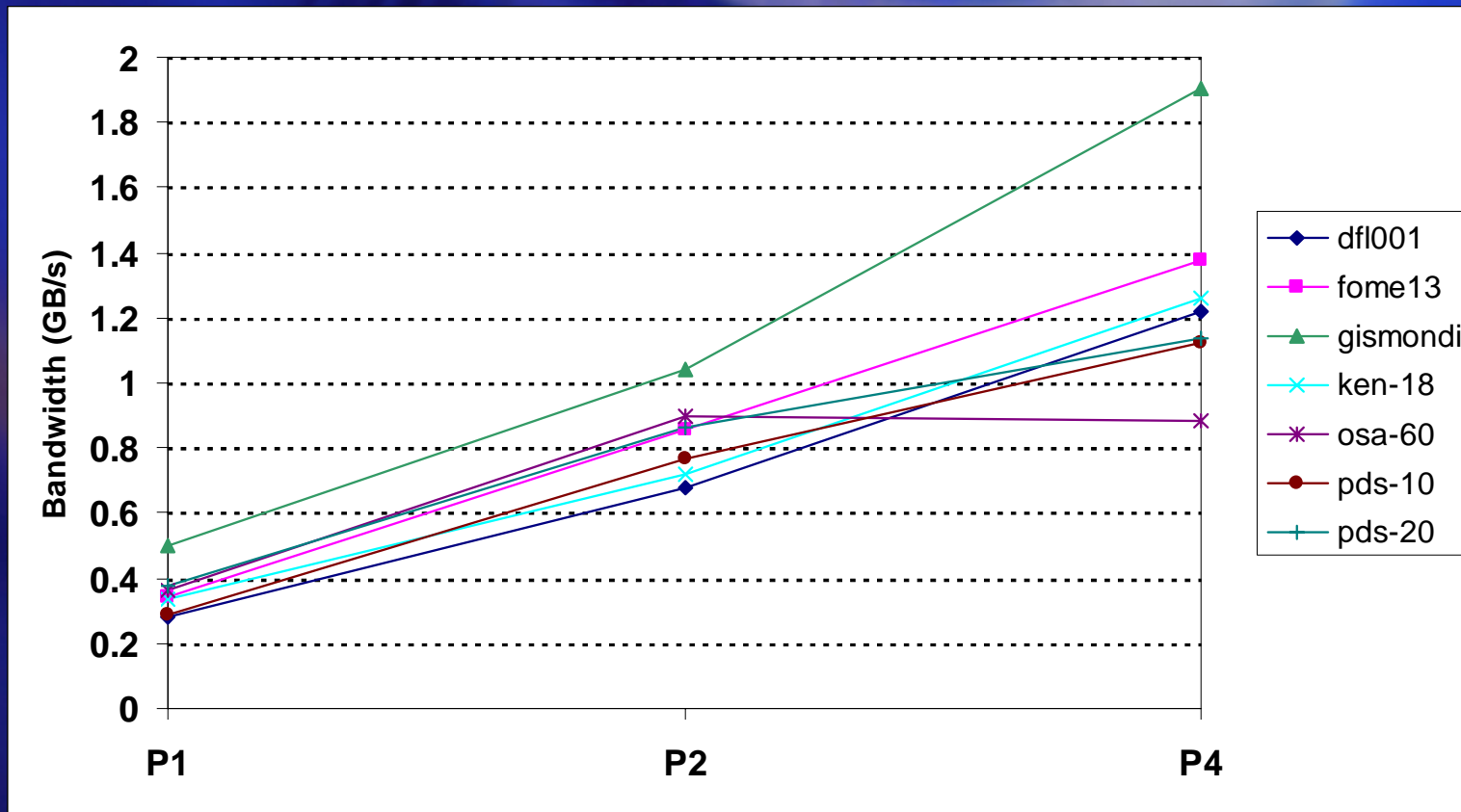
- L2 accesses are dominant (more than 75%)

Cache Behavior: Miss Rates



- There is good reuse from L2
- Problem size is too large to fit into L3 (miss rate > 13%)

Sustained Memory Bandwidth



- Bus not saturated: less than 25% utilization of 6GB/s bus

Conclusions

- **Build fast interior point system (IPS) – only 2.6x slower than CPLEX, industrial strength solver**
- **Up to 3x speed up on 4 processors**
 - **Good scalability on Cholesky factorization**
 - **Significant amount of time is spent in the backward/forward substitution which scales poorly**
- **Hardware resources do not limit scalability on our system**
 - **Good reuse from L2 cache**
 - **IPS is not memory bandwidth bound on our system**
- **Implementation of backward/forward substitution limits scalability**
 - **recent limit study reveals large potential speedup**
 - **code analysis shows inefficiencies in OpenMP implementation**
 - **current work targets to remove these inefficiencies**