Research goals:

- An application emulator based on Sparse Gaussian Elimination.
- Fast simulation of MPI programs on SMPs.
- Scheduling techniques for performance prediction and optimization.

Talk outline:

- Experience with high performance sparse Gaussian Elimination.
- Other ongoing research activities.

Performance Prediction and Optimization for Sparse Gaussian Elimination: A Case Study

Tao Yang
Department of Computer Science
University of California
Santa Barbara, CA 93106

Research work on sparse GE

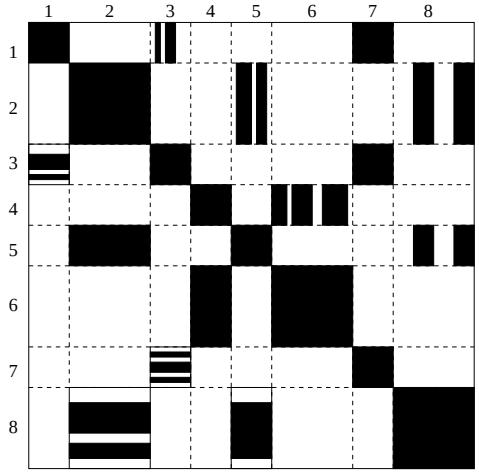
- Sequential machines.
  - UMFPACK (Davis and Duff. 93-94)
  - SuperLU (Demmel, Eisenstat, Gilbert, Li and Liu. 95-96)
- Shared memory machines.
  - SuperLU (Demmel, Gilbert, Li, 96-97). 2.5Gflops.
- Distributed memory machines.
  - $S*$ (Fu, Yang, SC'96). 1.3Gflops.
  - $S+$ (Shen, Jiao, Yang, SPAA'98). 11 Gflops.

Sparse Gaussian Elimination

A good case for us to study on performance prediction and optimization:

- Important for many applications.
- Challenges for high performance:
  - Poor caching performance due to compressed data representation and pivoting.
  - Limited parallelism.
  - Unpredictable dependency and data structures, processor loads.

Data partitioning after symbolic factorization

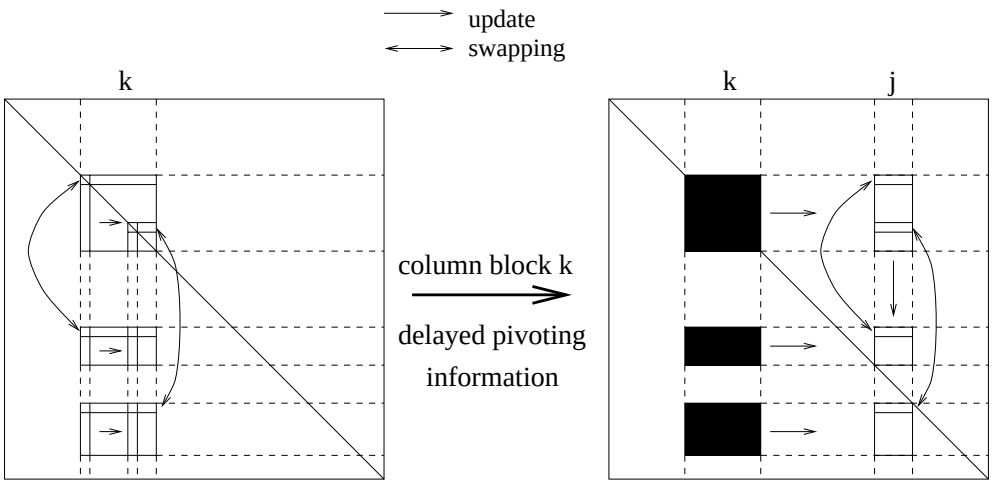


- Static preprocessing to fix data structure.
- Identify more dense structures to maximize the use of BLAS-3 (L/U supernode partitioning and amalgamation).

Our research strategies

- **Sequential complexity $x+1$ may be better than x.**
  Increase the operation count for simpler data structures and better caching performance.
  $\Longrightarrow$ Maximize the use of BLAS-3 and eliminate dynamic symbolic factorization.
- **Code prototyping.** Use the RAPID/PYRROS software for performance prediction and optimization.
- **Further optimization** by eliminating RAPID system overhead. Propose compact parallelism representation and simpler scheduling.

Task description



Factor(k): pivoting & local updating

Update(k,j): swapping and updating

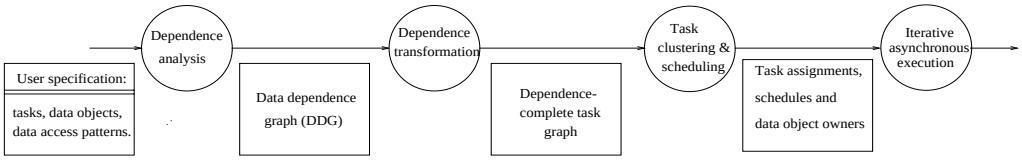
Program partitioning for sparse LU

```
(01) for k = 1 to N
(02)        Perform task Factor(k);
(03)        for j = k+1 to N with A_{k,j} ≠ 0
(04)             Perform task Update(k, j);
(05)        endfor
(06) endfor
```

- **Factor(k)**: Factorize and pivoting on column block k.
- **Update(k,j):**, Delayed updating/swapping that uses column block k to modify j.

Parallel solution using RAPID

RAPID provides an API and tool support for runtime parallelization of irregular codes with mixed granularities [ICS'96, PPoPP'97]:



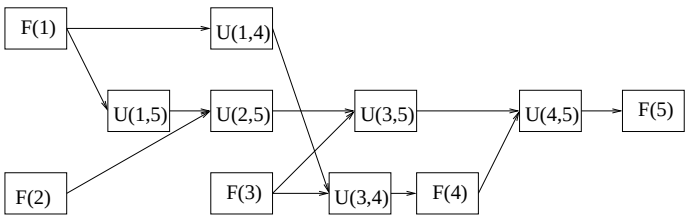
- User specifies shared data objects, tasks and data access patterns.
- RAPID derives a dependence graph and uses DAG scheduling (PYRROS) for mapping and performance prediction.

Task dependencies

Given the nonzero pattern of a matrix:

	1	2	3	4	5
1	1	0	0	1	1
2	0	1	0	0	1
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	1	1

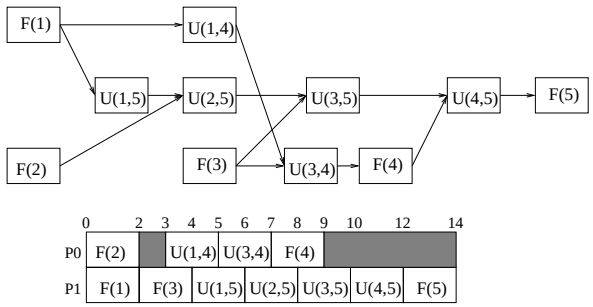
(a)



(b)

Scheduling for sparse GE

- **1D data mapping:** Column blocks are mapped to processors cyclically.
- **Task mapping:** Tasks that modify the same column block are in the same processor. Tasks are ordered within each processor.
- **Performance prediction:** Parallel time and space usage.



- RAPID provides an efficient schedule execution protocol using asynchronous fast remote memory access.
- Runs on Cray T3E, Meiko CS-2. Tested for sparse Cholesky/LU, sparse triangular solver, Newton's method, n-body simulation (fast multipole method).
- **Source code:** http://www.cs.ucsb.edu/research/rapid_sweb/RAPID.html.

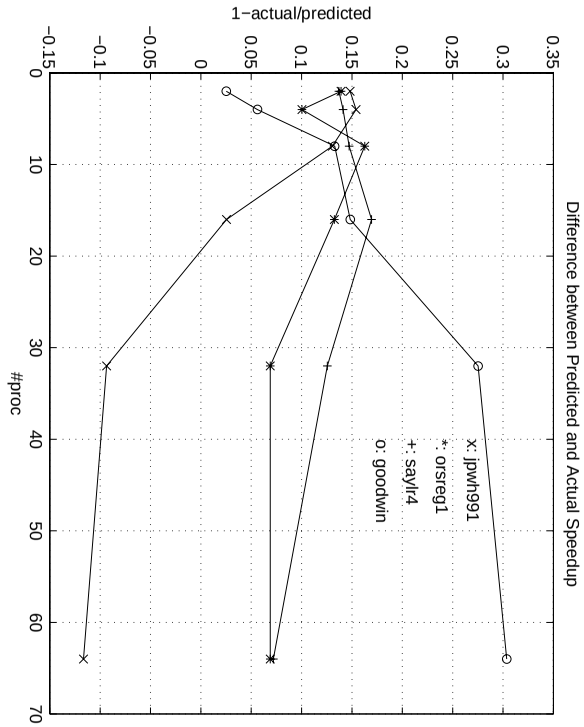
RAPID code on 300Mhz T3E

$$\text{Mflops} = \frac{\text{Operation count from SuperLU}}{\text{Parallel time of our code}}.$$

Matrix	P=2	P=8	P=16	P=32	P =64
sherman5	44.4	133	169.6	210.7	229.9
lnsp3937	34.3	94	145.5	183.5	201.0
lns3937	32.6	93	135.4	148.9	165.5
sherman3	51.4	144	192.8	199.0	212.7
jpwh991	41.4	124	173.9	193.2	217.3
orsreg1	53.4	160	215.6	223.3	231.6
goodwin	73.6	238	373.7	522.6	655.8
e40r0100	-	242	426.1	649.8	699.4
b33_5600	105.0	365	642.0	1029.0	1353.2

Effectiveness of performance prediction

Actual vs. predicted speedups:



Experience learned from RAPID prototyping

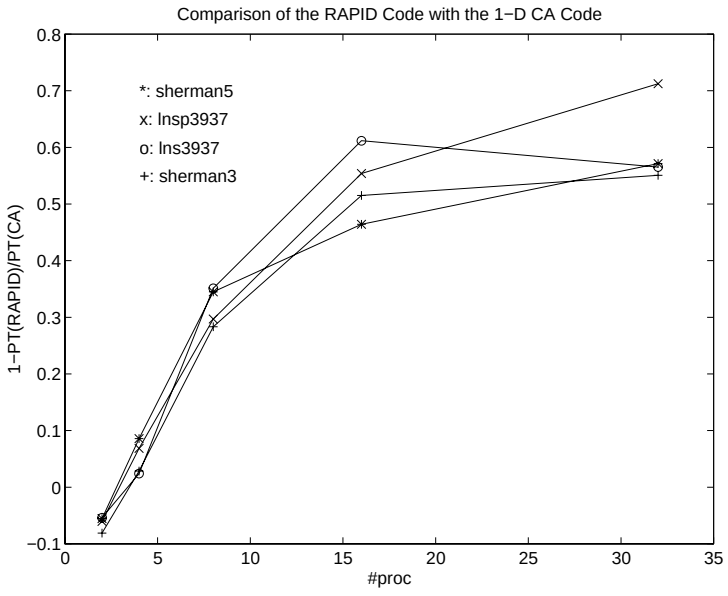
- Fast prototyping and reasonable performance prediction of parallel sparse LU code.
- Set a new performance record for distributed memory machines [SC'96].

RAPID can predict performance for solving larger matrices, but cannot execute schedules due to memory constraints.

- Predicted space usage is high, exceeding the memory capacity.
- Current RAPID system overhead is still substantial for runtime execution.

Effectiveness of graph scheduling

RAPID/PYRROS scheduling vs. a simpler method.



Overall sequential performance

Cray T3E. 300Mhz. Megaflop count does not include extra unnecessary operation introduced by static factorization.

Matrix	Sequential S^+		SuperLU		Sequential S^*		Exec. Time Ratio	
	Time	Mflops	Time	Mflops	Time	Mflops	$\frac{S^+}{SuperLU}$	$\frac{S^+}{S^*}$
sherman5	0.65	38.88	0.78	32.4	0.94	26.9	0.83	0.69
lnsp3937	1.48	28.52	1.73	24.4	2.0	21.1	0.86	0.74
lns3937	1.58	28.30	1.84	24.3	2.19	20.4	0.86	0.72
sherman3	1.56	39.52	1.68	36.7	2.03	30.4	0.93	0.77
jpwh991	0.52	33.38	0.56	31.0	0.69	25.2	0.93	0.75
orsreg1	1.60	39.11	1.53	40.9	2.04	30.7	1.05	0.78
saylr4	2.67	40.10	2.69	39.8	3.53	30.3	0.99	0.76
goodwin	10.26	65.28	-	-	17.0	39.4	-	0.60
dense1000	4.04	165.0	8.39	79.4	4.04	165.0	0.48	1.00

S^+: Optimized code for sparse LU

Key new ideas [SPAA'98].

- Supernodal matrix multiplication GEMM to improve the kernel computation speed.
- Compact parallelism representation: Elimination forest to guide
  - supernode partitioning and amalgamation.
  - 2D asynchronous computation scheduling with small space overhead.

Parallel performance of S^+

on 450MHz Cray T3E

Matrix	P=8		P=32		P=128	
	Time	Mflops	Time	Mflops	Time	Mflops
goodwin	1.21	553.5	0.69	970.6	0.67	999.6
e40r0100	4.06	611.3	1.87	1327.2	1.59	1560.9
raefsky4	38.62	804.2	11.54	2691.5	4.55	6826.0
inaccura	6.56	697.2	2.80	1633.4	1.91	2394.6
af23560	10.57	602.1	4.06	1567.5	2.80	2272.9
fidap011	21.58	1149.5	6.81	3642.7	3.04	8159.9
vavasis3	62.68	1423.6	19.26	4632.9	8.08	11043.5

Megaflop performance of S^+

on 300MHz Cray T3E

Matrix	P=8	P=16	P=32	P=64	P=128
goodwin	403.5	603.4	736.0	797.3	826.8
e40r0100	443.2	727.8	992.8	1204.8	1272.8
raefsky4	568.2	1072.5	1930.3	3398.1	5133.6
inaccura	495.5	803.8	1203.6	1627.6	1921.7
af23560	432.1	753.2	1161.3	1518.9	1844.7
fidap011	811.2	1522.8	2625.0	4247.6	6248.4
vavasis3	958.4	1864.8	3303.6	5640.4	8441.9

Megaflop rate of GEMM: 388.

Current work and research plans

- **Redesign and implement S+ using MPI as a performance benchmark** (application emulator).
- **Fast simulation of MPI codes on SMPs**
  - **Step 1:** Provide compiler and runtime support for thread-safe execution of MPI codes (ThrMPI).
  - **Step 2:** Integrate some of Howsim,Gigasim, Dumsim, Petasim, and PYRROS+ work for performance prediction without direct execution.

Summary on sparse GE:

- Effective time/space performance prediction and code prototyping by RAPID.
- Time and space efficient implementation for parallel sparse LU factorization on distributed memory machines.
- Achieve the highest megaflop rate. 11.04Gflops on 128-node 450MHz Cray T3E.

  Comparison: 2.5 Gflops by SuperLU on 8 processors of Cray C90.

Current ThrMPI performance

Performance improvement of ThrMPI over SGI MPI runtime system on a 4-processor Power Challenge.

Matrix size	4 MPI nodes	16 MPI nodes
MM (500 × 500)	32%	19%
MM (1000 × 1000)	32%	49%
GE (512 × 512)	4.8%	195.3%
GE (1024 × 1024)	same	163.1%

MM - Dense matrix multiplication. GE - Dense Gaussian Elimination.

- **Techniques for task graph scheduling and performance prediction:**
  - Continue to work on time/space performance prediction and optimization [PPoPP'97].
  - Symbolic scheduling and performance prediction for parameterized task graphs.
  - Integrate RAPID with PYRROS+, and use S* code as a test benchmark.