Nanoscale Integrated Circuits and Systems (NICS) Laboratory

Nonzero Pattern Analysis and Memory Access Optimization in GPU-based Sparse LU Factorization for Circuit Simulation

Xiaoming Chen, Du Su, Yu Wang, Huazhong Yang

Department of Electronic Engineering,

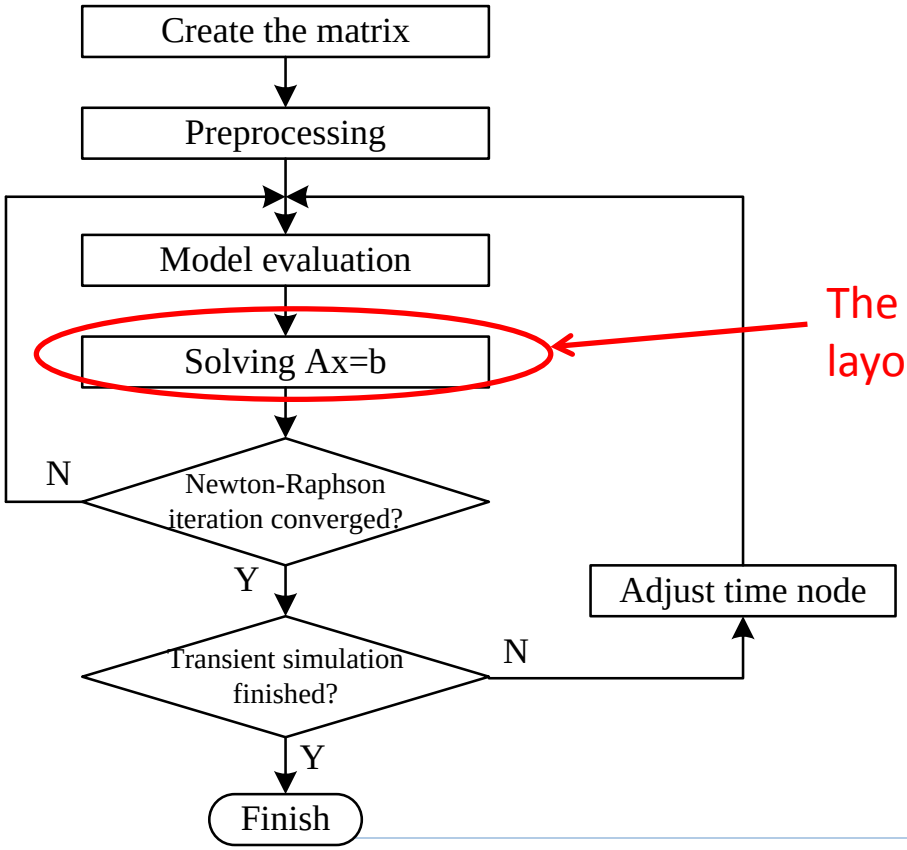
Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China

Outline

- Backgrounds & motivation
- Nonzero pattern & memory access analysis
- Blocked LU factorization on GPUs
- Results
- Conclusions

Backgrounds

- Simulation Program with Integrated Circuit Emphasis (SPICE) for IC simulation



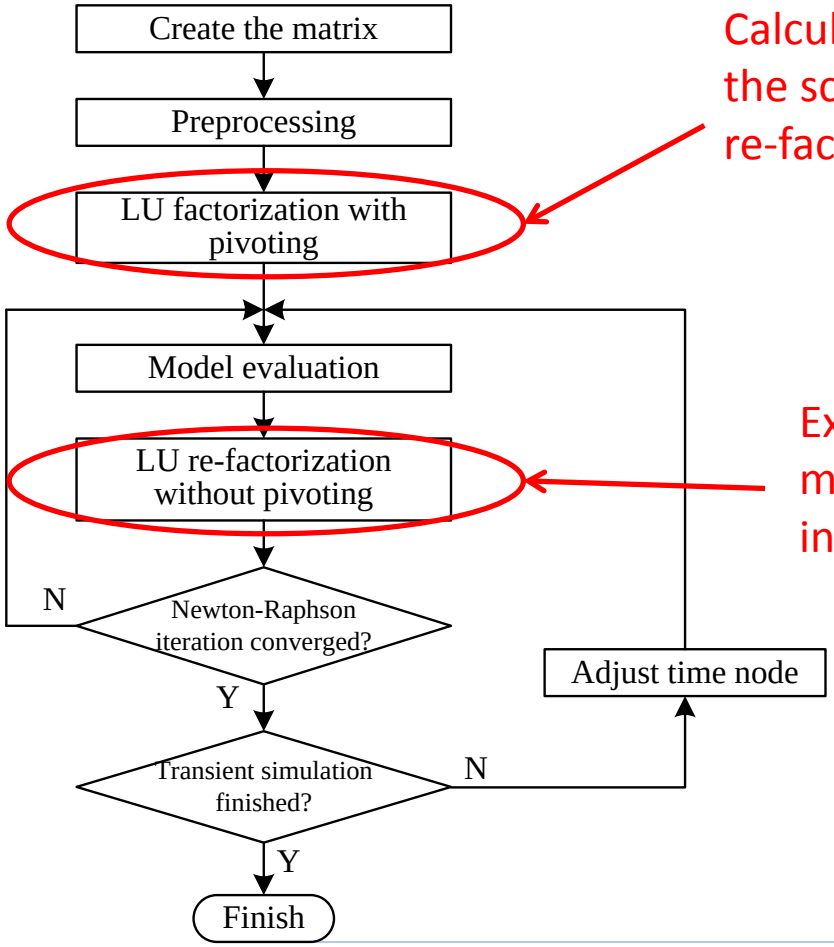
The most time-consuming step in post-layout simulation

Backgrounds

- Features of circuit matrices:
  - Irregular: asymmetric, possibly not positive definite
    - LU factorization is preferred
  - Highly sparse: nonzeros per row < 10
  - Fixed matrix structure during all iterations, only nonzero values change
    - Preparation steps (e.g. symbolic factorization) are executed just once
    - Numerical LU factorization is repeated for many times, with fixed matrix structure

Backgrounds

- A GPU-applicable flow [Ren, DAC'12]



Calculate the structure of the LU factors, the scheduling information for parallel LU re-factorization is also obtained

Executed on the GPU, using the fixed matrix structure and scheduling information

Backgrounds

- Work on GPU-based sparse LU factorization

reference	method	type/precision	average speedup/performance
Christen, GPPGPU'07	Supernodal with CUBLAS	Unsym/S	about 2X vs. sequential PARDISO, 12Gflop/s
Krawezik, SAAHPC'09	Multifrontal with BLAS3	Sym/D	2.9X vs. 2-threaded ANSYS
Yu, PC'11	Multifrontal with CUBLAS	Unsym/D	about 2.5X vs. sequential UMFPACK
George, IPDPS'11	Multifrontal with CUBLAS	Sym/S	7X vs. sequential (double-precision) WSMP 15X by 2 GPUs vs. sequential (double-precision) WSMP
Lucas, Tech. Rep. 2011	Multifrontal with CUBLAS	Sym/S	1.97X vs. sequential CPU code (their inhouse CPU code)
Lucas, VECPAR'10	Multifrontal with CUBLAS	Sym/S	5.91X vs. sequential CPU code (their inhouse CPU code) 1.34X vs. 8-threaded CPU code
Ren, DAC'12 (our previous work)	Left-looking no BLAS	Unsym/D	6.5X vs. sequential CPU code (our own solver) 1.14X vs. 8-threaded CPU code 0.91X vs. 10-threaded CPU code 10 Gflop/s

For general purposes

For circuit problems

Motivation

- BLAS-based methods are generally of lower performance than non-BLAS-based methods for circuit matrices [Davis, TOMS2010]
  - Circuit matrices are too sparse to form big supernodes
- Our previous work [Ren, DAC'12] targeting at circuit problems does not utilize BLAS
  - 10 Gflop/s (double-precision), achieving only 5% of the theoretical performance. However, a dense LU factorization achieves >80% of the theoretical performance [Tomov, IPDPSW'10]
  - There is still room for improvement

Contributions

- This work analyzed the nonzero patterns and memory access patterns in sparse LU factorization, to expose common features
- A blocked algorithm was studied to accelerate sparse LU factorization for circuit matrices, to accelerate circuit simulation

Nonzero pattern analysis

- Estimated density of nonzeros of the un-factorized matrix step by step

$$density(k) = \min\left\{\frac{2\,|E_k|+n-k}{(n-k)^2}, 1\right\}, k = 0,1,\cdots,n-1$$

$$|E_k| = \frac{-4\alpha^2|E_{k-1}|^3}{(n-k)(n-k+1)^3} + \frac{[2\alpha+2\alpha^2(n-k+1)]|E_{k-1}|^2}{(n-k)(n-k+1)^2} + \frac{[(n-k)^2-(3\alpha-1)(n-k)-\alpha]|E_{k-1}|}{(n-k)(n-k+1)}$$

Density of nonzeros of the un-factorized matrix will be higher and higher during factorization, finally it becomes a dense submatrix

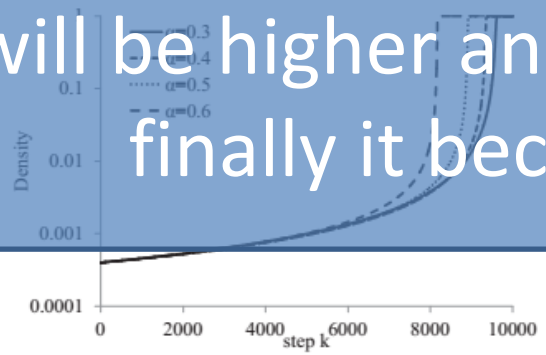


Figure 5: Trends of the density. $|V_0| = n = 10000, |E_0| = 15000$.

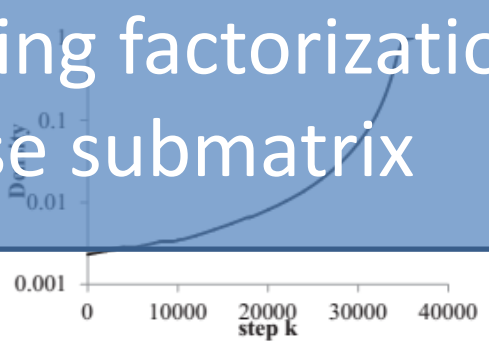


Figure 6: Trends of the density for factorizing onetone1 (36057×36057).

Memory access analysis

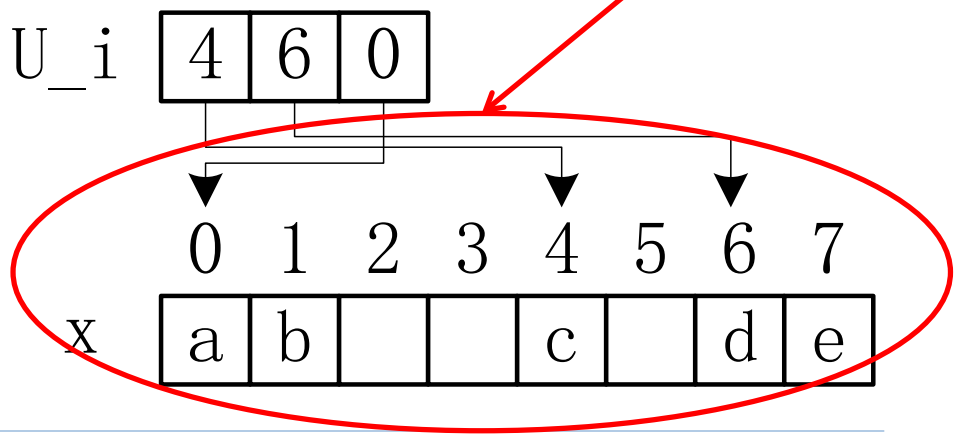
- Irregular memory accesses come from the sparse matrix structure

Algorithm 2 Detailed code for factorizing row i

```
1: //copy row i of A into a dense vector x
2: for (j=A_p[i]; j<A_p[i+1]; ++j)
3: {
4:   x[A_i[j]] = A_x[j];
5: }
6: //numeric accumulation from dependent rows
7: for (j=L_p[i]; j<L_p[i+1]; ++j)
8: {
9:   id = L_i[j];
10:  xj = x[id];
11:  for (k=U_p[id]; k<U_p[id+1]; ++k)
12:  {
13:    x[U_i[k]] -= xj*U_x[k];
14:  }
15: }
16: //storing factorization results
17: xj = x[i];
18: x[i] = 0.;
19: ldiag[i] = xj;
20: for (j=L_p[i]; j<L_p[i+1]; ++j)
21: {
22:   id = L_i[j];
23:   L_x[j] = x[id];
24:   x[id] = 0.;
25: }
26: for (j=U_p[i]; j<U_p[i+1]; ++j)
27: {
28:   id = U_i[j];
29:   U_x[j] = x[id] / xj;
30:   x[id] = 0.;
31: }
```

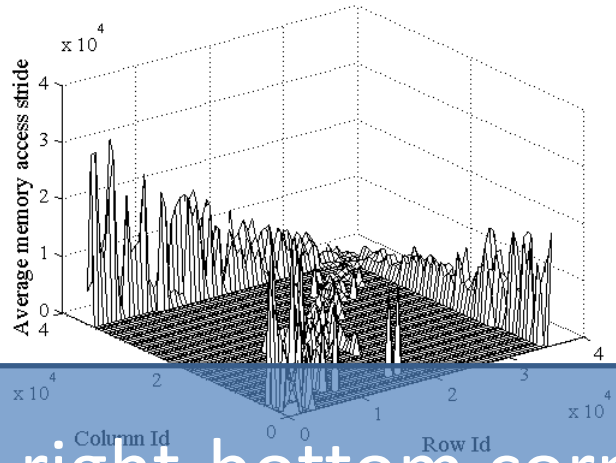
```
x[U_i[k]] -= xj*U_x[k];
```

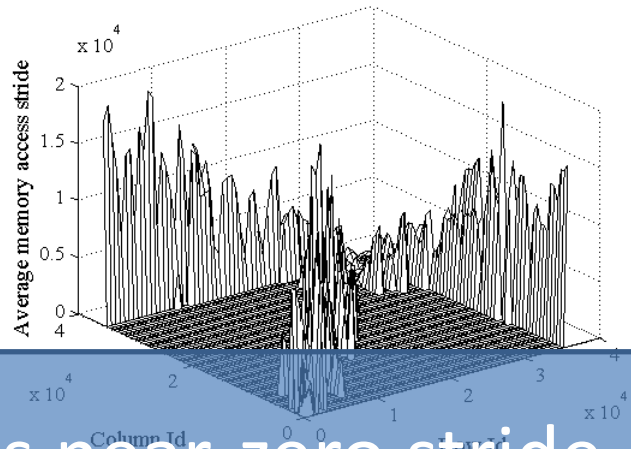
Indirect and strided memory accesses on the uncompressed array x, leading to ***un-coalesced*** memory accesses on GPUs



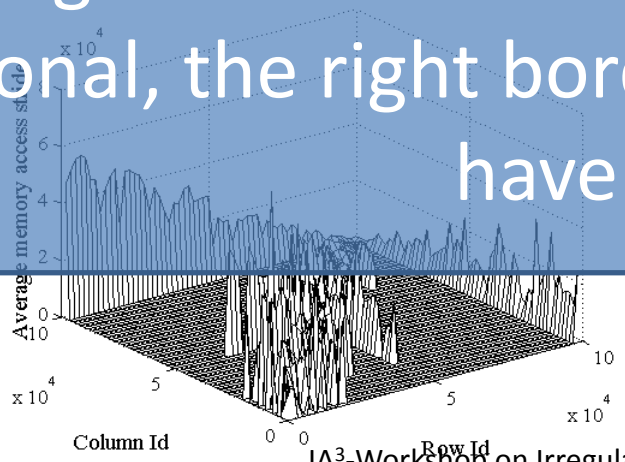
Memory access analysis

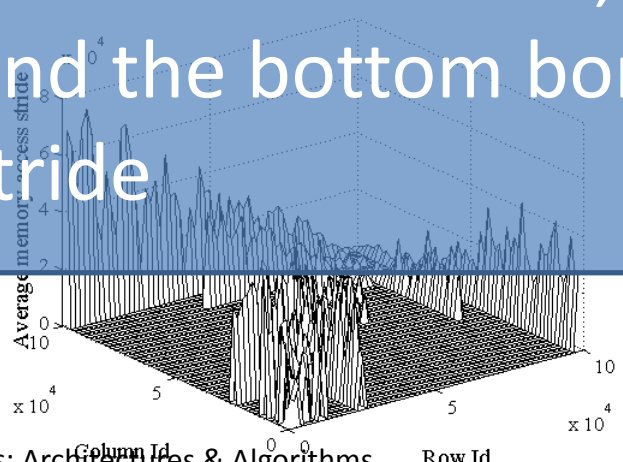
- Stride in memory accesses at each position





The right-bottom corner has near-zero stride, the diagonal, the right border, and the bottom border have big stride



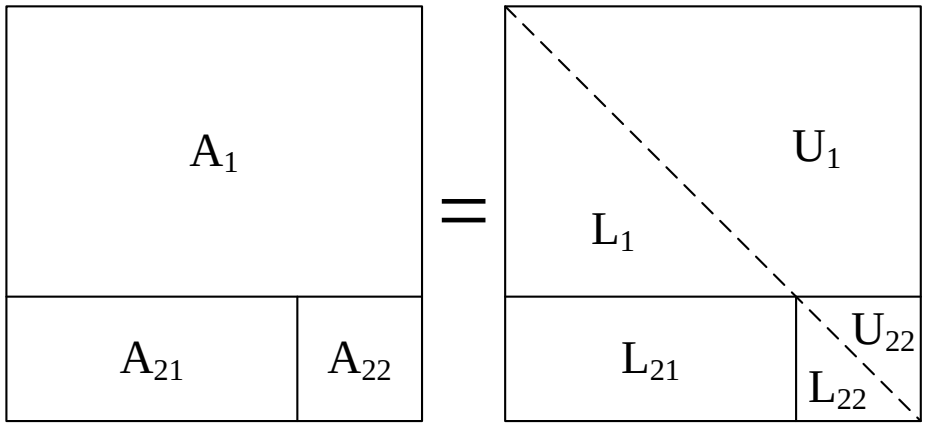


Analysis conclusions

- Observations
  - Sparse LU factorization should be separated into two parts to perform different optimization strategies
    - The former part has larger stride in memory access addresses, but fewer memory accesses
      - Optimization: sort the nonzeros of the LU factors in each row (proposed in [Ren, DAC'12], ~2X speedup)
    - The latter part has smaller stride but more memory accesses
      - There is a dense subblock at the right-bottom corner
      - Optimization: blocked method combining sparse and dense algorithms

Blocked LU factorization on GPUs

- Mathematically equivalent Schur-complement



$$\mathbf{A}_1 = \mathbf{L}_1\mathbf{U}_1$$ → partial factorization (ParFact)

$$\mathbf{L}_{21} = \mathbf{A}_{21}\mathbf{U}_1(:,1:p)^{-1}$$ → complement factorization (ComFact)

$$\mathbf{S} = \mathbf{A}_{22} - \mathbf{L}_{21}\mathbf{U}_1(:,p:n)$$ → sparse multiplication (SpMul)

$$\mathbf{S} = \mathbf{L}_{22}\mathbf{U}_{22}$$ → dense factorization (DenFact)

A method similar to [Tomov, IPDPSW'10]

Blocked LU factorization on GPUs

- Choice of the partition point

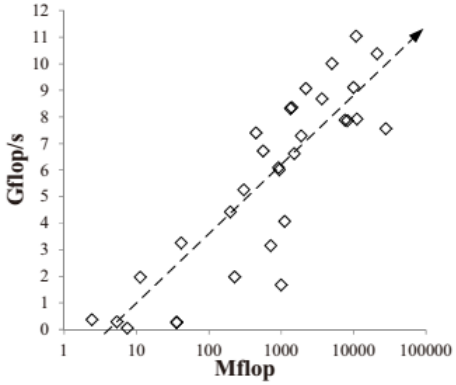


Figure 11: Performance of sparse LU factorization on NVIDIA GTX580 (double-precision). Mflop=million floating-point operations.

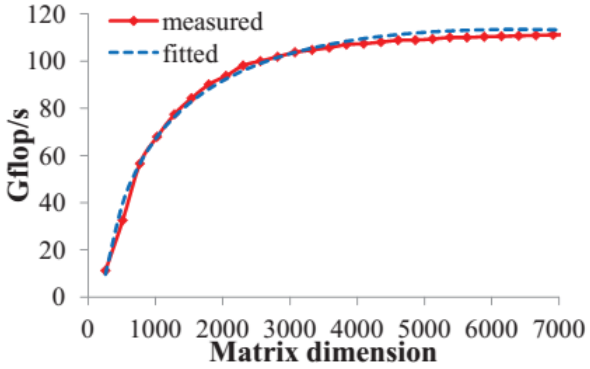


Figure 12: Performance of dense LU factorization on NVIDIA GTX580 (double-precision).

$$\text{sparse\_performance}(flop) = 2.535 \times log_{10}(Mflop) - 1.468$$

$$\text{dense\_performance}(k) = 31.93 \times log_2(k) - \frac{k}{138.23} - 243.92$$

$$\text{predicted factorization time} = \frac{\text{sparse\_flop}(p)}{\text{sparse\_performance(flop)}} + \frac{\text{dense\_flop}(n-p)}{\text{dense\_performance}(n-p)}$$

Experimental environment

- GPU: NVIDIA GeForce GTX580
  - Compute capability: ~200 Gflop/s (double-precision)
  - Off-chip bandwidth: 192.4 GB/s
- CPU: i7-3770K
  - 4 cores
- The blocked method is compared with the unblocked method and MKL PARDISO (4 threads)
- Benchmarks: from University of Florida Sparse Matrix Collection

Results

Table 1: Comparison between unblocked and blocked LU factorization.

Matrix	factorization time		Gflop/s		bandwidth (GB/s)		speedup	speedup vs. PARDISO	dense block dimension	padding %[a]
	unblocked	blocked	unblocked	blocked	unblocked	blocked				
circuit matrices										
asic_100k	0.273	0.235	4.07	4.73	57.2	66.4	1.16	0.79	1280	23.5
asic_100ks	0.166	0.094	8.38	14.86	117.6	208.6	1.77	1.01	1280	28.0
asic_320ks	0.168	0.168	8.13	8.13	114.2	114.2	1.00	3.51	256	98.9
asic_680ks	0.150	0.079	6.11	11.65	85.9	163.7	1.91	69.80	1408	38.3
ckt11752_dc_1	0.058	0.052	5.26	5.85	74.1	82.4	1.11	1.00	768	39.9
g2_circuit	1.094	0.778	9.17	12.89	128.6	180.8	1.41	0.47	2304	32.7
onetone1	0.163	0.065	8.24	20.54	115.6	288.1	2.49	2.22	1664	35.8
onetone2	0.045	0.026	4.42	7.49	62.3	105.4	1.69	2.02	1280	63.2
twotone	0.983	0.383	11.05	28.35	154.8	397.3	2.57	2.89	3584	40.3
average							**1.68**	**2.20**[b]		
non-circuit matrices										
zhao1	0.420	0.232	8.68	15.72	121.9	220.8	1.81	0.38	2048	36.9
sme3dc	1.403	1.107	7.92	10.03	111.2	140.9	1.27	0.39	2048	29.8
xenon1	2.034	1.561	10.38	13.53	145.7	189.8	1.30	0.21	2944	37.0
denormal	0.501	0.396	10.01	12.67	140.8	178.3	1.27	0.52	2048	39.5
thermomech_dm	0.260	0.257	7.29	7.37	103.3	104.4	1.01	0.72	896	45.1
thermomech_dk	0.970	0.850	7.88	8.99	111.0	126.6	1.14	0.34	1792	44.5
thermomech_tc	0.157	0.148	6.01	6.38	85.1	90.4	1.06	0.66	896	45.1
helm2d03	3.704	2.650	7.56	10.56	106.1	148.3	1.40	0.18	2432	18.8
average							**1.28**	**0.42**		

[a] percentage of the explicit zeros filled in the right-bottom dense block
[b] this value is the geometric mean, other average values are the arithmetic mean.

Results

- Performance and achieved bandwidth

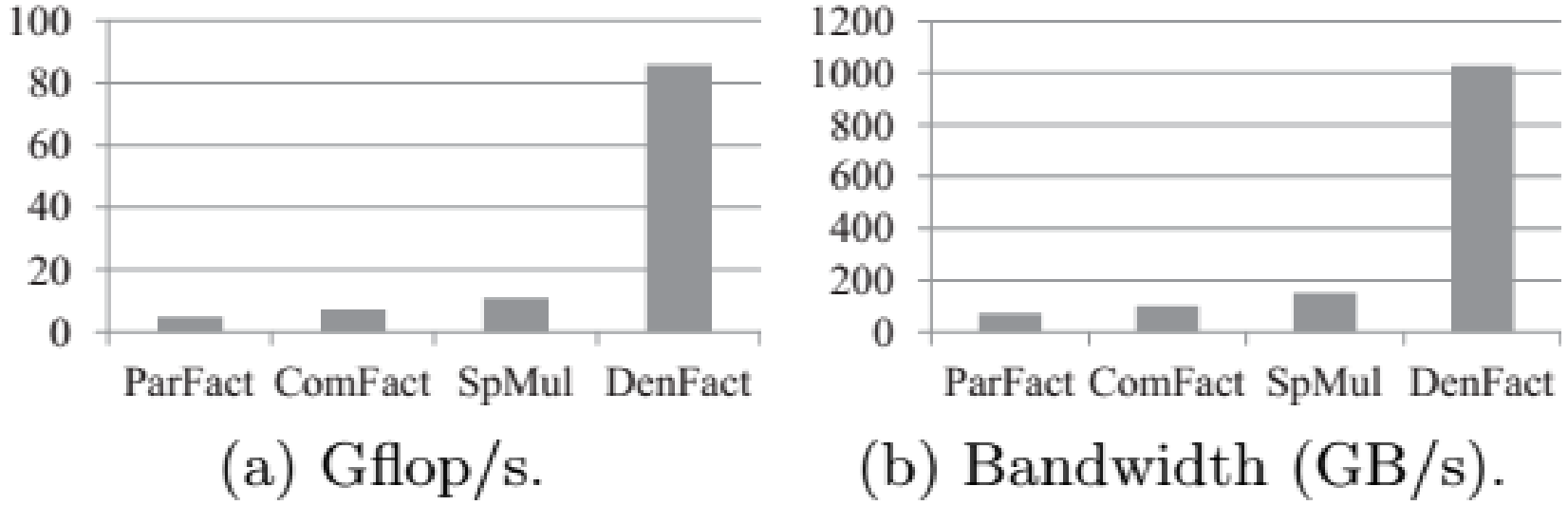


Figure 13: Gflop/s and bandwidth of each step in blocked LU factorization, for onetone1.

Conclusions

- Irregular matrix structure leads to irregular memory access patterns in sparse LU factorization
- We have proved that for circuit matrices which are highly sparse, a pure non-BLAS based method is not the best choice on GPUs
- A hybrid method combining sparse and dense algorithms is studied, achieving on average 68% improvement compared with the original method
- Our method is not conflict with BLAS-based methods
  - The sparse part can also use BLAS to achieve performance improvement for BLAS-based methods

Thanks for your attention
Q&A

For more questions, contact
chenxm05@mails.tsinghua.edu.cn