Performance Evaluation of Parallel Sparse Matrix-Vector Products on SGI Altix3700

H. Kotakemori [1], H. Hasegawa [2], T. Kajiyama [1]
A. Nukada [1], R. Suda [1], A. Nishida [1]

[1] University of Tokyo / CREST, JST
[2] University of Tsukuba / CREST, JST

Outline

- Introduction
- Sparse Matrix-Vector Product
- SGI Altix3700
  - NUMA architecture
  - First-touch mechanism
- Experiments
  - Sparse matrix-vector product
  - Conversion costs
- Conclusions

Introduction (1)

SSI
Scalable Software Infrastructure

- Demands for reliable and portable parallel numerical libraries are growing.
- Scalable Software Infrastructure Project
  - Started as a 5-year national projects since Nov. 2002.
  - Development
    - Portable implementation of the following libraries:
    - Parallel eigen solvers
    - Parallel linear system solvers
    - Parallel fast integral transforms

Introduction (2)

- We are planning to develop a library of iterative solvers, which includes a wide range of iterative solvers, preconditioners, and storage formats.
- The matrix-vector product is the most important kernel operation for iterative linear solvers.
- Its performance has a significant effect on the performance of linear solvers.

Introduction (3)

- We discuss the performance of sparse matrix-vector products on a cc-NUMA machine SGI Altix3700.
- What’s problems :
  - First-touch mechanism
  - The performance of sparse matrix-vector product for each storage format.
  - conversion costs of the storage format.

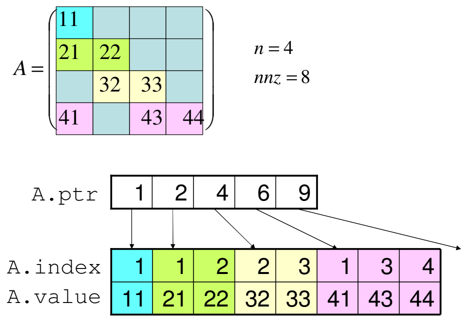
Outline

- Introduction
- Sparse Matrix-Vector Product
- SGI Altix3700
  - NUMA architecture
  - First-touch mechanism
- Experiments
  - Sparse matrix-vector product
  - Conversion costs
- Conclusions

Sparse Matrix-Vector Product with OpenMP

- Sparse Matrix-Vector Product y=Ax
  - The storage formats affect the performance
- Parallelize using OpenMP.
  - OpenMP is designed for shared memory machines.
- Advantages
  - a serial program can be parallelized one loop at a time.
  - Compiler directives are used, so that the same code can be compiled for serial or parallel execution.
  - portability
- Special treatment for data locality, such as first-touch, may be required, especially for cc-NUMA architectures (will be discussed later).

Compressed Row Storage (CRS)



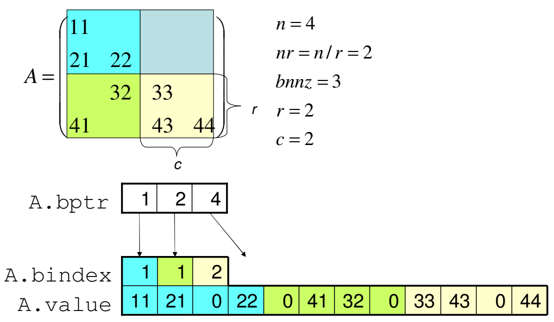
Matrix-Vector Product for CRS

```
for(i=0; i<n; i++)  {
  t = 0.0;
  for(j=A.ptr[i];j<A.ptr[i+1];j++)
    t += A.value[j] * x[A.index[j]];
  y[i] = t;
}
```

Matrix-Vector Product for CRS with OpenMP

```
#pragma omp parallel for private(i,j,t)
  for(i=0; i<n; i++)  {
    t = 0.0;
    for(j=A.ptr[i];j<A.ptr[i+1];j++)
      t += A.value[j] * x[A.index[j]];
    y[i] = t;
  }
```

Block Sparse Row (BSR)



Matrix-Vecotr Product for BSR

```
for(i=0; i<nr; i++)  {
  t0 = t1 = 0.0;
  for(j=A.bptr[i];j<A.bptr[i+1];j++)  {
    jj  = A.bindex[j];
    t0 += A.value[j*4+0] * x[jj*2+0];
    t1 += A.value[j*4+1] * x[jj*2+0];
    t0 += A.value[j*4+2] * x[jj*2+1];
    t1 += A.value[j*4+3] * x[jj*2+1];
  }
  y[2*i+0] = t0; y[2*i+1] = t1;
}
```

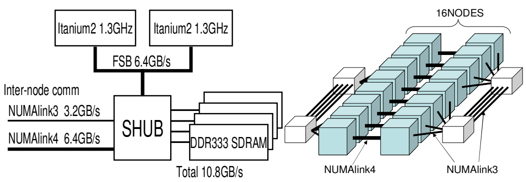
Matrix-Vecotr Product for BSR with OpenMP

```
#pragma omp parallel for private(i,j,jj,t0,t1)
  for(i=0; i<nr; i++)  {
    t0 = t1 = 0.0;
    for(j=A.bptr[i];j<A.bptr[i+1];j++)  {
      jj  = A.bindex[j];
      t0 += A.value[j*4+0] * x[jj*2+0];
      t1 += A.value[j*4+1] * x[jj*2+0];
      t0 += A.value[j*4+2] * x[jj*2+1];
      t1 += A.value[j*4+3] * x[jj*2+1];
    }
    y[2*i+0] = t0; y[2*i+1] = t1;
  }
```

Outline

- Introduction
- Sparse Matrix-Vector Product
- SGI Altix3700
  - NUMA architecture
  - First-touch mechanism
- Experiments
  - Sparse matrix-vector product
  - Conversion costs
- Conclusions

NUMA Architecture



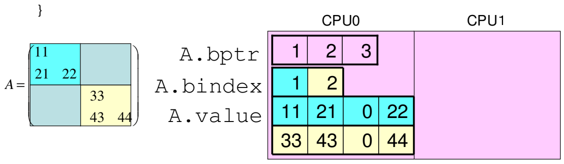
- With 16 or fewer threads, the threads are allocated to different nodes using the `dplace` command.
- With 32 processors, the bus of each node is shared with the two processors in the node.

First-touch Mechanism

- Each page is stored in the memory of the node with a processor that accesses the page first.
- Data must be transferred via interconnects when it is accessed by a processor out of the node that owns the data.
- It is necessary to take into account the first-touch mechanism for the construction of each storage format.

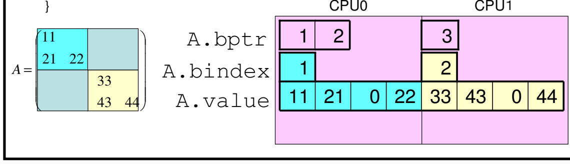
Convert from CRS to BSR (Sequential)

```
for(bi=0;bi<nr;bi++)  {
  i  = bi*r; ii = 0;
  while( i+ii<n && ii<=r-1 )  {
    for( k=Ain.ptr[i+ii];k<Ain.ptr[i+ii+1];k++)  {
      ......
        Aout.bindex[kk] = Ain.index[k]/c;     Aout.value[ij]  = Ain.value[k];   kk = kk+1;
      ......      }
    ii = ii+1;
  }
  Aout.bptr[bi] = kk;
  ......
}
```

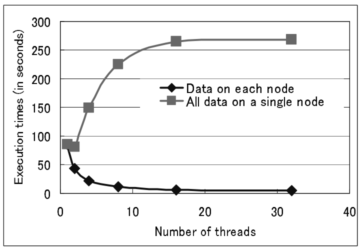


Convert from CRS to BSR (Parallel)

```
#pragma omp parallel for private(...)
 for(bi=0;bi<nr;bi++)  {
  i  = bi*r;   ii = 0;    kk = Aout.bptr[bi];
  while( i+ii<n && ii<=r-1 )  {
    for( k=Ain.ptr[i+ii];k<Ain.ptr[i+ii+1];k++)  {
      ......
        Aout.bindex[kk] = Ain.index[k]/c;  Aout.value[ij]  = Ain.value[k];   kk = kk+1;
      ......
    }
    ii = ii+1;
  }
  ......
}
```



Control First-touch vs. NOT Control First-touch



- All data on a single node is poor performance.
- The data distribution is important for taking into account the first-touch mechanism.

Summary : SGI Altix3700

- In order to obtain good performance, each page should be assigned to the node with the processor that most often accesses the page.
- To control first-touch, we parallelized the storage format conversion routines.

Outline

- Introduction
- Sparse Matrix-Vector Product
- SGI Altix3700
  - NUMA architecture
  - First-touch mechanism
- Experiments
  - Sparse matrix-vector product
  - Conversion costs
- Conclusions

Experiments

- We examined
  - times of parallel matrix-vector products
  - speed-ups of parallel matrix-vector products
  - storage format conversion costs

Test Matrices

	Name	Dimension	Nonzeros	Ave.
(a)	af23560	23,560	484,256	20.55
(b)	fidapm37	9,152	765,944	83.69
(c)	fidap011	16,614	1,091,362	65.69
(d)	bcsstk30	28,924	2,043,492	70.65
(e)	s3dkq4m2	90,449	4,820,891	53.30
(f)	Poisson	1,000,000	26,463,592	26.46

- (a) to (e) : Matrix Market.
- (f) : FEM of the three-dimensional Poisson equation on a cube.
- Ave : The average number of the non-zero elements per row.

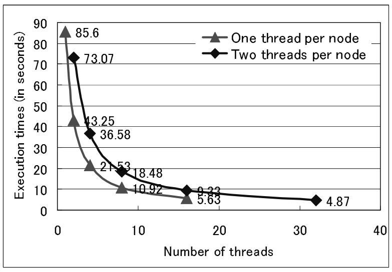
Execution times (in seconds) of 1000 iterations

Number of threads		1	2	4	8	16	32
Matrix	Format						
(a)	CRS	3.79	1.89	0.91	0.46	0.24	0.14
	BSR_41	**1.46**	**0.72**	**0.28**	**0.15**	**0.09**	**0.07**
(b)	CRS	2.53	1.33	0.63	0.32	0.18	0.10
	BSR_22	**2.24**	**1.19**	**0.57**	**0.24**	**0.14**	**0.09**
(c)	CRS	3.87	1.98	1.01	0.48	0.26	0.15
	BSR_41	**2.51**	**1.30**	**0.65**	**0.24**	**0.13**	**0.09**
(d)	CRS	6.81	3.53	1.88	0.97	0.46	0.24
	BSR_41	**4.48**	**2.34**	**1.30**	**0.61**	**0.23**	**0.14**
(e)	CRS	20.87	10.47	5.26	2.71	1.43	0.68
	BSR_41	**9.17**	**4.65**	**2.39**	**1.30**	**0.62**	**0.27**
(f)	CRS	149.50	74.96	37.43	18.76	9.51	4.97
	BSR_31	**85.60**	**43.25**	**21.53**	**10.92**	5.63	4.87
	DIA	178.50	89.19	44.34	16.40	**4.72**	**2.81**

Speed-up ratios

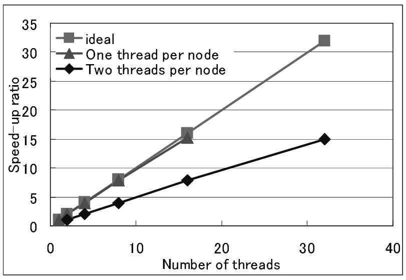
Number of threads		1	2	4	8	16	32
Matrix	Format						
(a)	CRS	1.00	2.00	4.18	8.19	15.51	27.16
	BSR_41	1.00	2.04	5.19	9.59	15.77	21.69
(b)	CRS	1.00	1.90	3.99	7.93	14.23	24.14
	BSR_22	1.00	1.88	3.91	9.42	15.90	25.18
(c)	CRS	1.00	1.95	3.82	8.03	15.13	26.50
	BSR_41	1.00	1.93	3.83	10.60	18.75	28.03
(d)	CRS	1.00	1.93	3.63	7.00	14.91	28.07
	BSR_41	1.00	1.91	3.45	7.34	19.22	32.21
(e)	CRS	1.00	1.99	3.97	7.70	14.61	30.72
	BSR_41	1.00	1.97	3.83	7.04	14.73	33.63
(f)	CRS	1.00	1.99	3.99	7.97	15.72	30.07
	BSR_31	1.00	1.98	3.97	7.84	15.20	17.58
	DIA	1.00	2.00	4.03	10.88	37.84	63.51

Result of BSR_31 for matrix (f)



- The absolute performance for the two threads per node is lower than the one thread per node.

Result of BSR_31 for matrix (f)



- The speed-ups relative to the performance with two threads are steady up to 32 threads.

Summary : Sparse matrix-vector product

- The speed-ups have attained good results for any storage format when the FSB was dedicated to one CPU.
- The performance for the BSR format causes a great decrease when the FSB is shared with two CPUs.
- The cache and memory bus architectures have been observed to influence the optimum choice of the storage format.

Conversion Costs

- Assumptions:
  - T_{crs} : the execution times of MV in the CRS formats.
  - T_{tgt} : the execution times of MV target formats.
  - T_{conv} : the execution times of the conversion from the CRS format to the target format.
- Conversion Costs

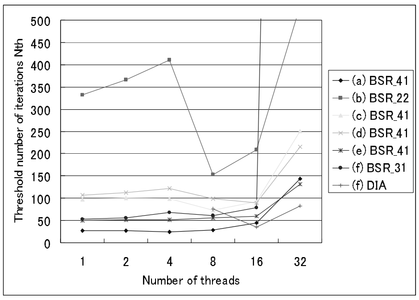
$$N_{th} = \left\lceil \frac{T_{conv}}{T_{crs} - T_{tgt}} \right\rceil$$

  - If the number of MV ≧ Nth then it is better to use the target format; otherwise it is better to use CRS format without conversion.

Conversion times T_{conv} (in milliseconds)

Number of threads		1	2	4	8	16	32
Matrix	Format						
(a)	BSR_41	61.2	30.7	15.0	8.5	6.7	10.4
(b)	BSR_22	96.9	50.8	24.9	12.4	7.7	8.5
(c)	BSR_41	132.8	68.1	35.4	17.8	11.1	14.1
(d)	BSR_41	247.6	132.3	69.8	35.9	20.2	22.2
(e)	BSR_41	575.9	292.7	148.5	78.2	47.7	53.5
(f)	BSR_31	3370.8	1720.3	1073.5	478.6	303.8	439.2
	DIA	907.4	485.6	270.3	178.0	165.7	178.8

Threshold numbers of iterations N_{th}



Summary : Conversion Costs

- The value of Nth changes slightly except (b).
- The conversion of the storage format provides faster computation of the matrix-vector product
  - If the number of the matrix-vector product is 100 times or more in this test matrices.

Conclusions (1)

- Our Implementations have attained satisfactory scalability.
  - It is necessary to take into account the first-touch mechanism.
- The storage format has been observed to greatly affect the performance of matrix-vector products.
  - In order to maximize the performance of a machine, users must be able to choose an appropriate storage format for each matrix.
- The conversion of the storage format provides faster computation of the matrix-vector product
  - If the number of the matrix-vector product is certain times or more.

Conclusions (2)

- To take into account the First-touch mechanism.
  - we parallelized the storage format conversion routines using OpenMP.

Future Works

- We are planning to port and to evaluate our codes to other shared memory parallel machines.
- Our next goal is parallelization for distributed memory parallel machines through MPI and MPI-OpenMP hybrid parallelization.
- We will also work toward high-performance iterative linear solvers using these kernel routines and effective preconditioners for the solvers.

Acknowledgements

- This research was supported in part by CREST “Development of Software Infrastructure for Large Scale Scientific Simulation”, Japan Science and Technology Agency.