Multidimensional Blocking in UPC

Christopher Barton, Călin Caşcaval, George Almási,
Rahul Garg, José Nelson Amaral, Montse Farreras

October 11, 2007

© 2007 IBM Corporation

Overview

- **Current data distribution in UPC**
- **Multiblock extension to UPC**
- **Locality analysis**
- **Results**
- **Conclusions and Future Work**

 © 2007 IBM Corporation

Unified Parallel C

- **Parallel extension to C**
  - Allow the declaration of shared variables
  - Programmer specifies shared data distribution
  - Work-sharing construct used to distribute loop iterations
  - Synchronization primitives: barriers, fences, and locks
- **Data can be private, shared local and shared remote**
  - Shared data can be accessed by all threads
  - Shared space is divided into portions with affinity to a thread
  - Latency to access *local* shared data is usually lower than latency to access *remote* shared data
- **All threads execute the same program (SPMD style)**
- **Convenient programming model for clusters of SMP**

© 2007 IBM Corporation

Data Distributions in UPC

- **Data distributions are an essential part of a PGAS language**
- **Affinity directives are already in the language**
  - shared [B] int A[N]
- **This distribution is not particularly well-suited for applications in specific domains**
  - Does not allow easy interface with existing high performance libraries

 © 2007 IBM Corporation

Multidimensional Blocking in UPC

- **Extend shared data layout directive to support multidimensional distributions**
- **Programmer specifies n-dimensional *tiles* for shared data**

```
shared [2][2] int X[10][10];
```

New Blocking factor → [2][2]

Array size → [10][10]

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

```
#define BLK_SIZE (sizeof(double)*b*b)
shared [b][b] double A[M][P], B[P][N], C[M][N];

upc_forall(int ii = 0; ii < M; ii += BS; continue)
   upc_forall(int jj = 0l jj < N; jj += BS; &C[ii][jj]) {
      double scratchA[b][b], scratchB[b][b];

      upc_memget(scratchA, &A[ii][jj], BLK_SIZE);
      upc_memget(scratchB, &B[ii][jj], BLK_SIZE);

      dgemm(&C[ii][jj], &scratchA, &scratchB, b, b, b);
   }
```


© 2007 IBM Corporation

Drawbacks to Multidimensional Blocking

- **Accessing shared data becomes more expensive**
  - Computing indexes requires sum and products across each shared dimension
- **Compiler can identify accesses to *local* shared data and privatize them**
  - Remove the overhead of calls to the runtime library
  - Expose the indexing computations to the compiler so they can be optimized

 © 2007 IBM Corporation

Locality Analysis

Goal

Compute symbolically the node ID of each shared reference in the upc_forall loop and compare it to the node ID of the affinity expression.

```
upc_forall (int i=0; i < N; i++; &A[i])
```

- **Shared reference inside the loop has a displacement with respect to the affinity expression**
  - Specified by the distance vector **k** = $[k_0, k_1, \ldots, k_{n-1}]$

© 2007 IBM Corporation

Locality Analysis

- **Observation 1**
  - A block shifted by a distance vector **k** can span *at most* two threads in each dimension
  - Locality can change in only one place in this dimension, which we call the *cut*
- **Observation 2**
  - Only need to test the locality at the 2^n corners of the block
  - All elements from the *corner* to the *cut* will have the same owner

© 2007 IBM Corporation

Identifying Local Shared Accesses

- **For a given upc_forall loop**
  - Split the loop into multiple iteration spaces where the shared reference is either local or remote
    - Use cuts to determine split points
  - For each iteration space, keep a *position* relative to the affinity test
    - Local shared references are privatized
    - Remote shared references are transformed into calls to the UPC Runtime

 © 2007 IBM Corporation

Computing Cuts

```
/*8 threads and 2 threads/node */
shared [2][3] int A[8][9];
```

```
for(v0=0; v0 <7 ; v0++) {
   upc_forall(v1=0; v1<6; v1++; &A[v0][v1]){
      A[v0+1][v1+2] = v0*v1;
   }
}
```

k =

[1,2]

	0	1	2	3	4	5	6	7	8
0	T_0	T_0	T_0	T_1	T_1	T_1	T_2	T_2	T_2
1	T_0	T_0	T_0	T_1	T_1	T_1	T_2	T_2	T_2
2	T_3	T_3	T_3	T_4	T_4	T_4	T_5	T_5	T_5
3	T_3	T_3	T_3	T_4	T_4	T_4	T_5	T_5	T_5
4	T_6	T_6	T_6	T_7	T_7	T_7	T_0	T_0	T_0
5	T_6	T_6	T_6	T_7	T_7	T_7	T_0	T_0	T_0
6	T_1	T_1	T_1	T_2	T_2	T_2	T_3	T_3	T_3
7	T_1	T_1	T_1	T_2	T_2	T_2	T_3	T_3	T_3

© 2007 IBM Corporation

Computing Cuts

```
/*8 threads and 2 threads/node */
shared [2][3] int A[8][9];
```

```
for(v0=0; v0 <7 ; v0++) {
   upc_forall(v1=0; v1<6; v1++; &A[v0][v1]){
      A[v0+1][v1+2] = v0*v1;
   }
}
```

k =
[1,2]

	0	1	2	3	4	5	6	7	8
0	T_0	T_0	T_0	T_1	T_1	T_1	T_2	T_2	T_2
1	T_0	T_0	T_0	T_1	T_1	T_1	T_2	T_2	T_2
2	T_3	T_3	T_3	T_4	T_4	T_4	T_5	T_5	T_5
3	T_3	T_3	T_3	T_4	T_4	T_4	T_5	T_5	T_5
4	T_6	T_6	T_6	T_7	T_7	T_7	T_0	T_0	T_0
5	T_6	T_6	T_6	T_7	T_7	T_7	T_0	T_0	T_0
6	T_1	T_1	T_1	T_2	T_2	T_2	T_3	T_3	T_3
7	T_1	T_1	T_1	T_2	T_2	T_2	T_3	T_3	T_3

0 0 1
1 2 2
3 3 0
0 1 1

Dimensions 0 to *n*-2

$Cut_i = b_i - k_i \% b_i$

$Cut_0 = 2 - 1 \% 2$

$Cut_0 = 1$

© 2007 IBM Corporation

Computing Cuts

```
/*8 threads and 2 threads/node */
shared [2][3] int A[8][9];
```

```
for(v0=0; v0 <7 ; v0++) {
  upc_forall(v1=0; v1<6; v1++; &A[v0][v1]){
    A[v0+1][v1+2] = v0*v1;
  }
}
```

k =

k' = $[k_0+b_0-1,k1]$

k' = [2,2]

	0	1	2	3	4	5	6	7	8
0	T_0	T_0	T_0	T_1	T_1	T_1	T_2	T_2	T_2
1	T_0	T_0	T_0	T_1	T_1	T_1	T_2	T_2	T_2
2	T_3	T_3	T_3	T_4	T_4	T_4	T_5	T_5	T_5
3	T_3	T_3	T_3	T_4	T_4	T_4	T_5	T_5	T_5
4	T_6	T_6	T_6	T_7	T_7	T_7	T_0	T_0	T_0
5	T_6	T_6	T_6	T_7	T_7	T_7	T_0	T_0	T_0
6	T_1	T_1	T_1	T_2	T_2	T_2	T_3	T_3	T_3
7	T_1	T_1	T_1	T_2	T_2	T_2	T_3	T_3	T_3

Block numbers: 0 0 1 / 1 2 2 / 3 3 0 / 0 1 1

Dimension *n*-1

$Cut^{Upper} = (tpn - course(k)) x b_{n-1} - k_{n-1} \% b_{n-1}$

$Cut^{Upper} = (2 - 0)x3 - 2 \% 3$

$Cut^{Upper} = 4$

$Cut^{Lower} = (tpn - course(k')) x b_{n-1} - k_{n-1} \% b_{n-1}$

$Cut^{Lower} = (2 - 1) x 3 - 2 \% 3$

$Cut^{Lower} = 1$

© 2007 IBM Corporation

8 UPC Threads
2 Nodes
4 Threads per node

Algorithm – Phase 1: Collection

```
1 shared [5][5] int A[20][20];
2 for (i=0; i < 19; i++)
3   upc_forall (j=0; j < 20; j++; &A[i][j]) {
4     A[i+1][j] = MYTHREAD;
5   }
```

	0	1	2	3	4	5	...
0	T_0	T_0	T_0	T_0	T_0	T_1	T_1
1	T_0	T_0	T_0	T_0	T_0	T_1	T_1
2	T_0	T_0	T_0	T_0	T_0	T_1	T_1
3	T_0	T_0	T_0	T_0	T_0	T_1	T_1
4	T_0	T_0	T_0	T_0	T_0	T_1	T_1
5	T_4	T_4	T_4	T_4	T_4	T_5	T_5
...	T_4	T_4	T_4	T_4	T_4	T_5	T_5

- **Collect shared references in upc_forall loop**
- **Compute distance vector**

A[i+1][j] = MYTHREAD; [1,0]

 © 2007 IBM Corporation

Phase 2: Restructure Loop

```
1 shared [5][5] int A[20][20];
2 for (i=0; i < 19; i++)
3   upc_forall (j=0; j < 20; j++; &A[i][j]) {
4     A[i+1][j] = MYTHREAD;
5   }
```

8 UPC Threads
2 Nodes
4 Threads per node

Distance Vector:
[1,0]

	0	1	2	3	4	5	...
0	T_0	T_0	T_0	T_0	T_0	T_1	T_1
1	T_0	T_0	T_0	T_0	T_0	T_1	T_1
2	T_0	T_0	T_0	T_0	T_0	T_1	T_1
3	T_0	T_0	T_0	T_0	T_0	T_1	T_1
4	T_0	T_0	T_0	T_0	T_0	T_1	T_1
5	T_4	T_4	T_4	T_4	T_4	T_5	T_5
...	T_4	T_4	T_4	T_4	T_4	T_5	T_5

- **Starting at outer loop, iteratively compute cuts for each loop**
- **Outer Loop:**

$Cut_i = b_i - k_i \% b_i$

$Cut_0 = 4$

 © 2007 IBM Corporation

Phase 2: Restructure Loop

```
1  shared [5][5] int A[20][20];
2  for (i=0; i < 19; i++)
3    if ( (i%5) < 4) {
4       upc_forall (j=0; j < 20; j++; &A[i][j]) {
5         A[i+1][j] = MYTHREAD;
6        }
7     }
8     else {
9       upc_forall (j=0; j < 20; j++; &A[i][j]) {
10        A[i+1][j] = MYTHREAD;
11       }
12    }
```

8 UPC Threads

2 Nodes

4 Threads per node

Distance Vector:

[1,0]

	0	1	2	3	4	5	...
0	T_0	T_0	T_0	T_0	T_0	T_1	T_1
1	T_0	T_0	T_0	T_0	T_0	T_1	T_1
2	T_0	T_0	T_0	T_0	T_0	T_1	T_1
3	T_0	T_0	T_0	T_0	T_0	T_1	T_1
4	T_0	T_0	T_0	T_0	T_0	T_1	T_1
5	T_4	T_4	T_4	T_4	T_4	T_5	T_5
...	T_4	T_4	T_4	T_4	T_4	T_5	T_5

- **Inner Loops:**
  - Offset is 0 so Cut^{Upper} and Cut^{Lower} are 0

© 2007 IBM Corporation

Phase 3: Privatize Local Accesses

```
1  shared [5][5] int A[20][20];
2  for (i=0; i < 19; i++)
3    if ( (i%5) < 4) {
4      upc_forall (j=0; j < 20; j++; &A[i][j]) {
5        A[i+1][j] = MYTHREAD;
6      }
7    }
8    else {
9      upc_forall (j=0; j < 20; j++; &A[i][j]) {
10       A[i+1][j] = MYTHREAD;
11     }
12   }
```

8 UPC Threads

2 Nodes

4 Threads per node

Distance Vector:

[1,0]

Region Position: [0,0]
Reference position: [1,0]
Node ID: 0 (Local)

Region Position: [4,0]
Reference Position: [5,0]
Node ID: 1 (Remote)

	0	1	2	3	4	5	...
0	T_0	T_0	T_0	T_0	T_0	T_1	T_1
1	T_0	T_0	T_0	T_0	T_0	T_1	T_1
2	T_0	T_0	T_0	T_0	T_0	T_1	T_1
3	T_0	T_0	T_0	T_0	T_0	T_1	T_1
4	T_0	T_0	T_0	T_0	T_0	T_1	T_1
5	T_4	T_4	T_4	T_4	T_4	T_5	T_5
...	T_4	T_4	T_4	T_4	T_4	T_5	T_5

© 2007 IBM Corporation

Phase 3: Privatize Local Accesses

8 UPC Threads

2 Nodes

4 Threads per node

Distance Vector:

[1,0]

```
1  shared [5][5] int A[20][20];
2  for (i=0; i < 19; i++)
3    if ( (i%5) < 4) {
4      upc_forall (j=0; j < 20; j++; &A[i][j]) {
5        offset = ComputeOffset(i,j);
6        *(base_A+offset) = MYTHREAD;
7      }
9    }
9    else {
10     upc_forall (j=0; j < 20; j++; &A[i][j]) {
11      A[i+1][j] = MYTHREAD;
12     }
13   }
```

	0	1	2	3	4	5	...
0	T_0	T_0	T_0	T_0	T_0	T_1	T_1
1	T_0	T_0	T_0	T_0	T_0	T_1	T_1
2	T_0	T_0	T_0	T_0	T_0	T_1	T_1
3	T_0	T_0	T_0	T_0	T_0	T_1	T_1
4	T_0	T_0	T_0	T_0	T_0	T_1	T_1
5	T_4	T_4	T_4	T_4	T_4	T_5	T_5
...	T_4	T_4	T_4	T_4	T_4	T_5	T_5

© 2007 IBM Corporation

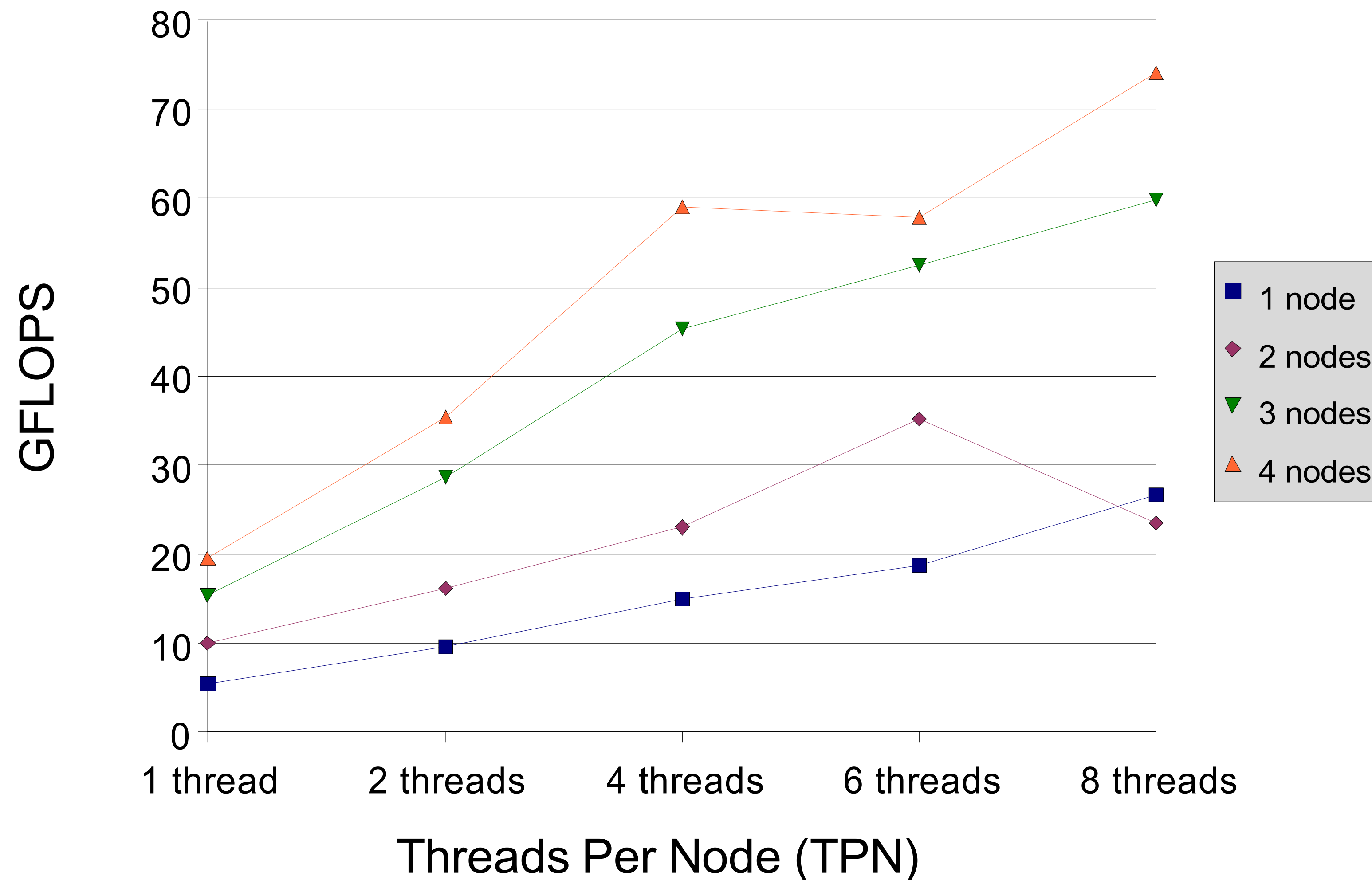
Preliminary Results

- **Machine**
  - 4 nodes of an development IBM Squadron Cluster
    - Each node: 8 SMP POWER5 processors, 1.9GHz; 16GB of memory
- **Benchmarks**
  - Cholesky factorization and Matrix Multiply
    - Showcase multiblocking for shared arrays
    - Implementation uses ESSL library for multiplying matrices
  - Dense matrix-vector multiply and 5-point stencil
    - Demonstrate performance impact of locality analysis and privatization

© 2007 IBM Corporation

Cholesky Factorization

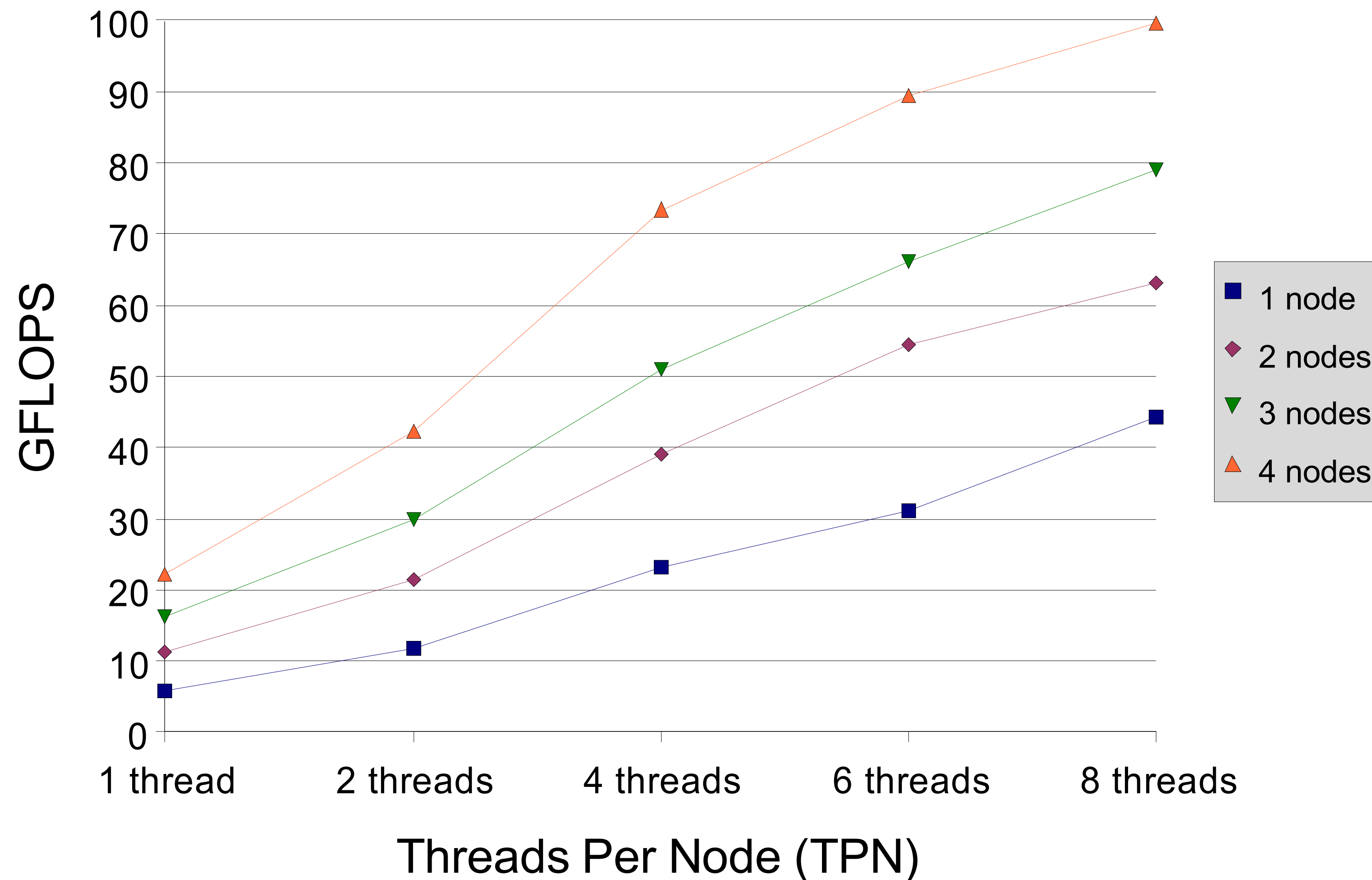
Theoretical Peak: 6.9GFlops x TPN x nodes



© 2007 IBM Corporation

Matrix Multiplication

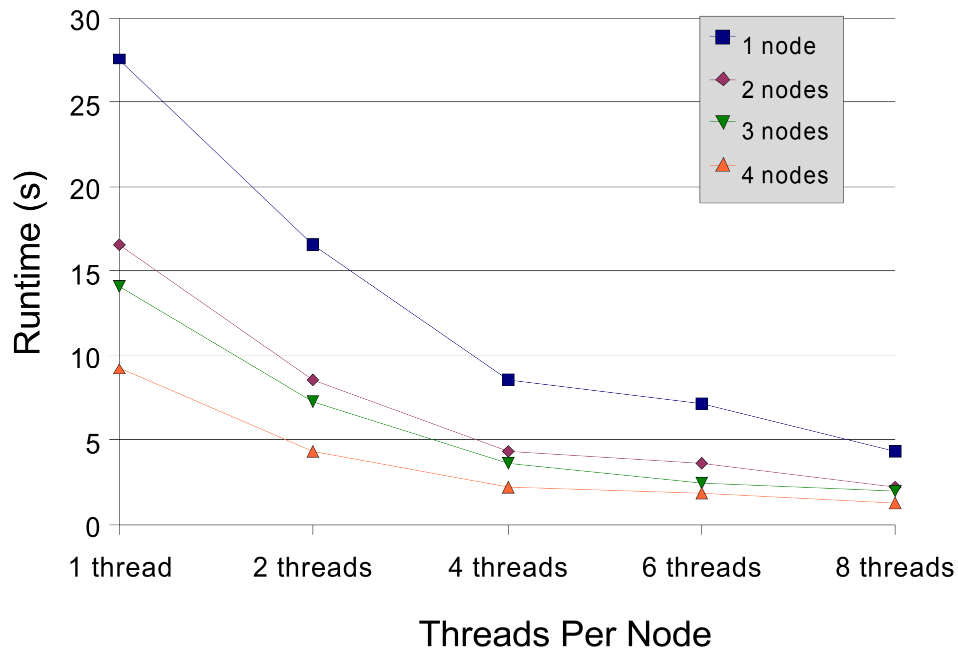
Theoretical Peak: 6.9GFlops x TPN x nodes

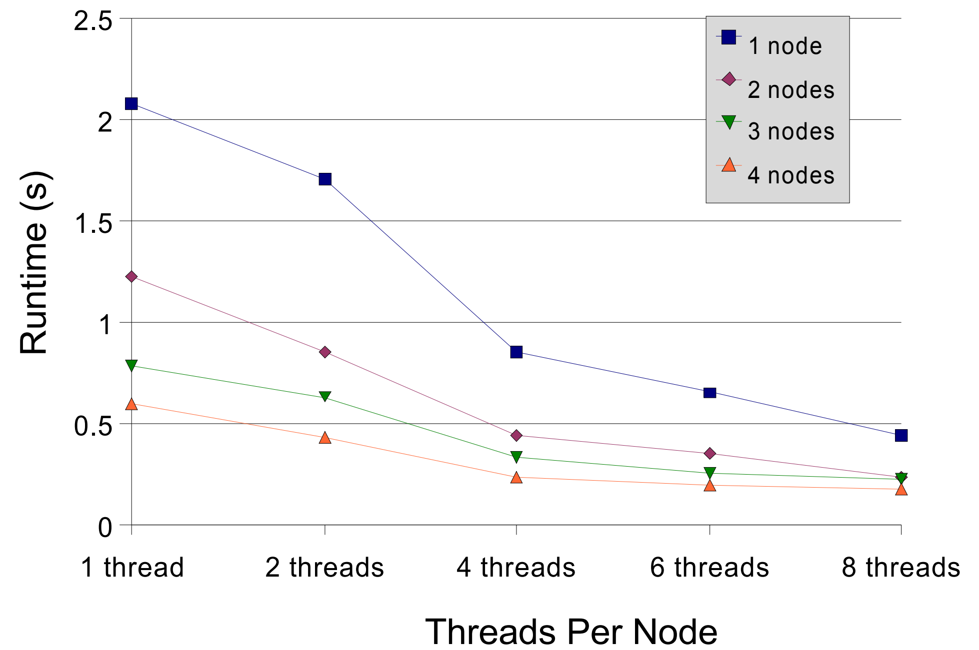


 © 2007 IBM Corporation

Locality Analysis and Privatization

Sequential C Time: 0.47s



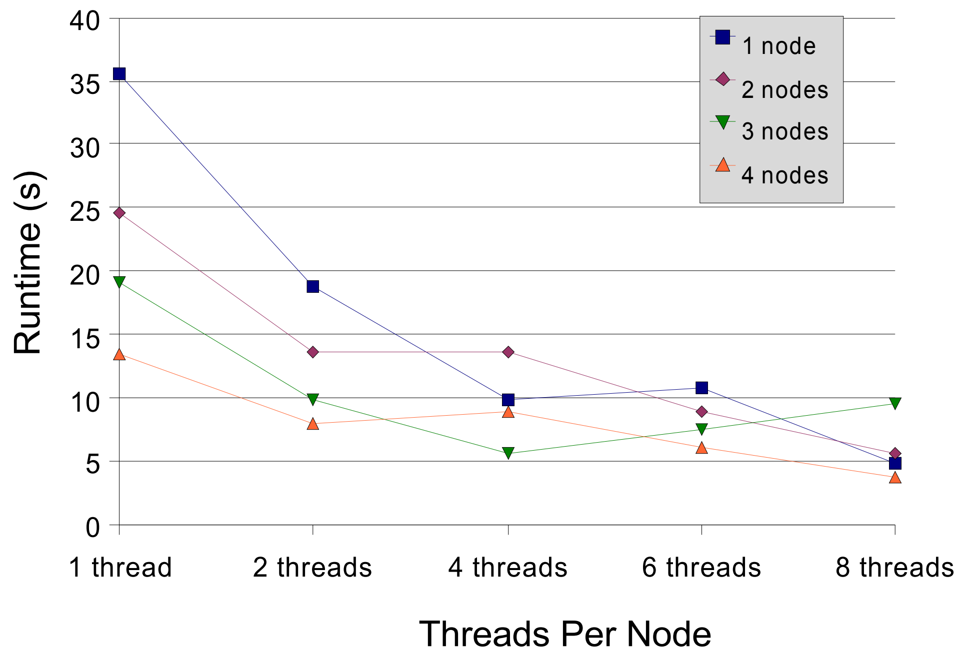


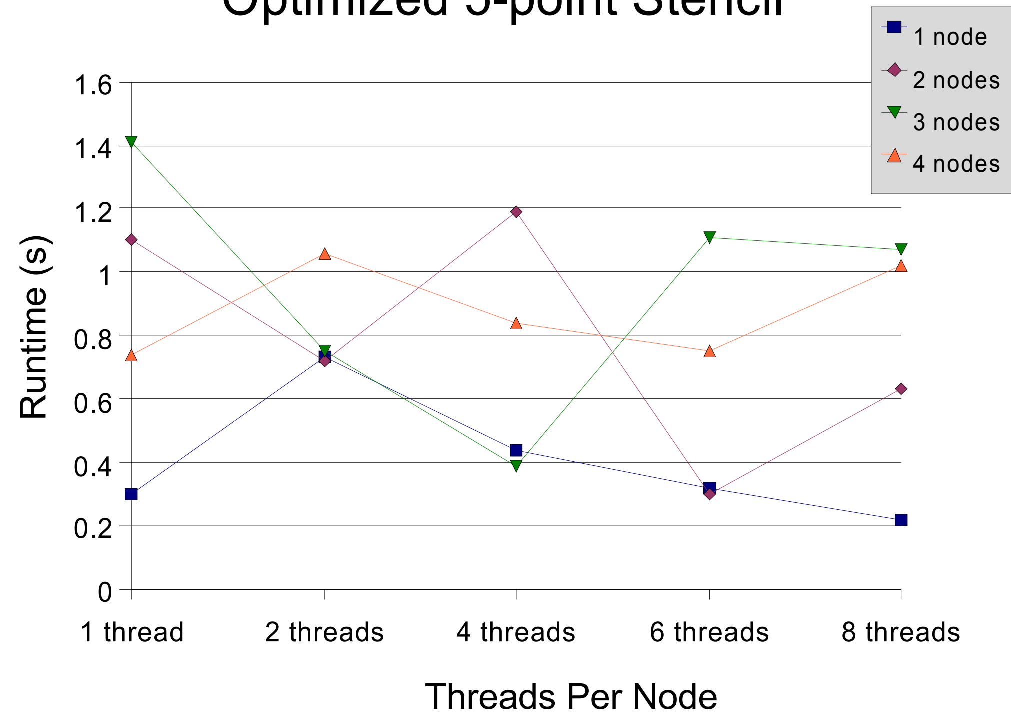

```
shared [*] double A[14400][14400];
```


© 2007 IBM Corporation

Locality Analysis and Privatization

Sequential C Time: 0.17s






```
shared [100][100] double A[6000][6000];
```

 © 2007 IBM Corporation

Conclusions

- **Multiblocked arrays fine control over data layout**
  - Allows programmer to obtain better performance while simplifying expression of computations
  - Provides ability to integrate with existing libraries (C and Fortran) that require specific memory layouts
- **Locality analysis**
  - Compiler is able to reduce the overheads introduced by accessing shared arrays

 © 2007 IBM Corporation

Future Work

- **Multiblocked arrays**
  - Add processor tiling to increase programmer's ability to write code that scales to a large number of processors
  - Define a set of collectives that are optimized for the UPC programming model that will address scalability issues
- **Compiler**
  - Reduce overhead of computing indexes of privatized accesses
  - Investigate how traditional loop optimizations can be used with upc_forall loops to further improve performance

© 2007 IBM Corporation

Questions

© 2007 IBM Corporation