



IBM TJ Watson Research Center

Productivity: The Quest for Performance without Pain

Calin Cascaval
cascaval@us.ibm.com

Overview

- **Programming productivity**
- **PGAS languages**
- **Dwarfs and PGAS**
- **Productivity enhancements for UPC**
- **Conclusions**

What is Programmer Productivity?

■ # of lines of code

- Pros: readability and maintainability
- Cons: software engineering, portability, and optimization

```
for(ii = 0; ii < N; ii+=B)
  for(jj = 0; jj < N; jj+=B)
    for (i = ii; i < MIN(N,ii+B); i++)
      for(j = jj; j < MIN(N,jj+B); j++) {
        C[i][j] = 0;
        for(k = 0; k < N; k++)
          C[i][j] += A[i][k]*B[k][j];
      }
}
```

MATLAB

$C = A * B$

What is Programmer Productivity?

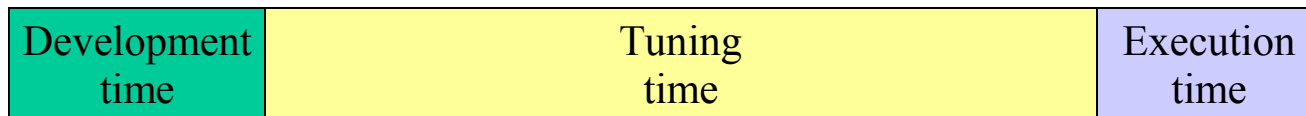
- **Language and runtime support: Garbage Collection**
 - Advantages
 - programmers do not need to do memory management
 - opened an entire area of research on garbage collection algorithms
 - Disadvantages
 - memory is a shared resource and garbage collection is inherently more wasteful than hand-coded allocation and deallocation
 - programmers become careless
- **But it is good enough for many**

The Performance Tuning Dilemma

Initial solution



Tuned solution



- **Tuning code can improve performance by a significant factor (5× or more) on modern architectures, hence **cannot be ignored****
- **Performance tuning requires significant level of system expertise, which is a **critical skill****
- **Tuning time can be significantly more than development time, thereby **reducing programmer productivity****

PGAS languages

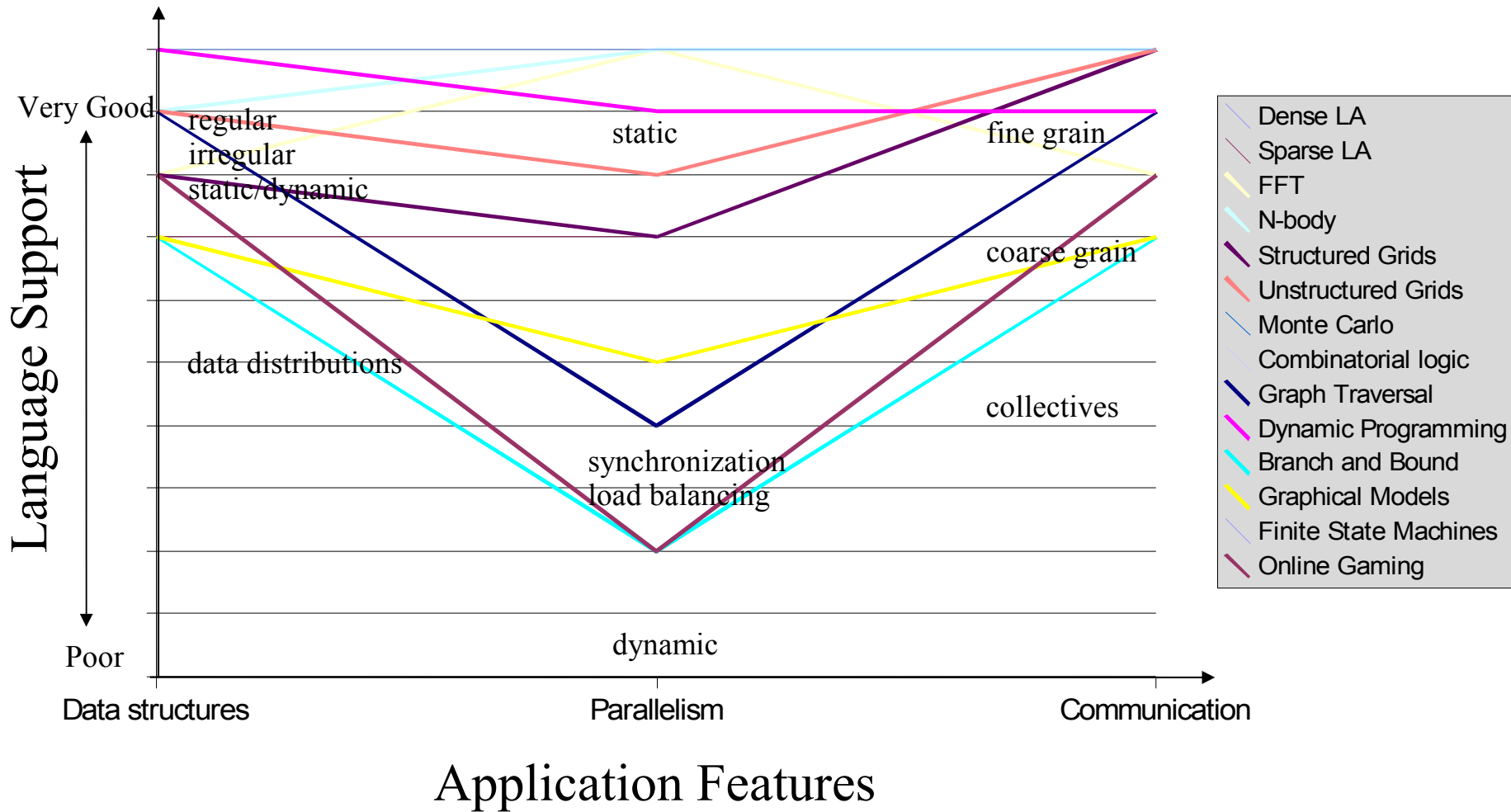
■ Features

- Small set of data parallel primitives typically grafted on an existing language: Co-Array Fortran, UPC, Titanium
- Shared memory-like programming with locality awareness – shared data is explicitly declared and distributions are implicit in the declaration
- SPMD threading model with synchronization primitives (barriers, fences, and locks)
- Collective communication and parallel I/O through libraries

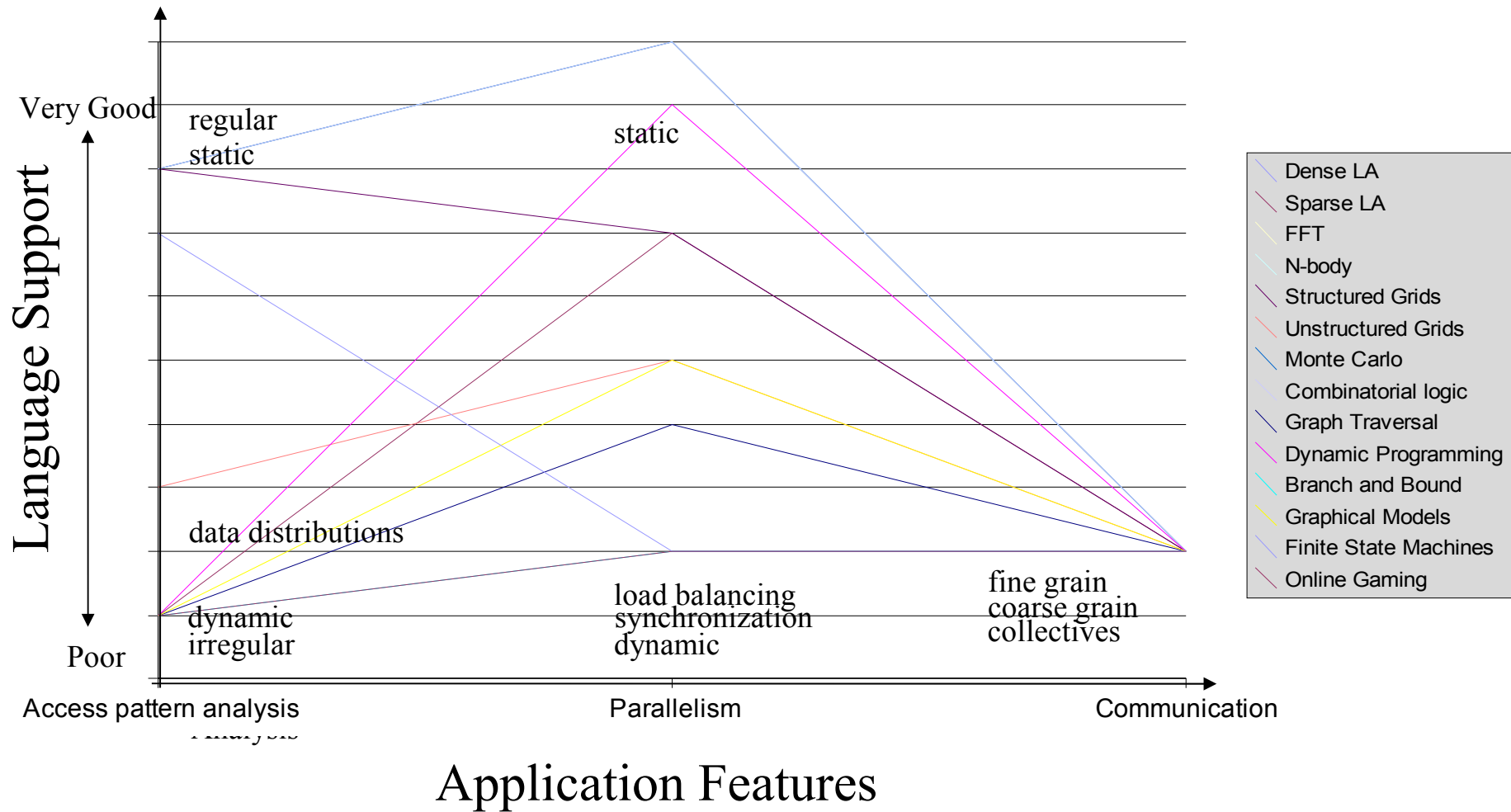
■ Implementation

- Can be mapped to shared memory, distributed memory and combinations (clusters of SMPs)
- One-sided communication

UPC – Language Expressivity



UPC – Performance through Compiler Analysis



UPC Performance Gaps

- **Efficient single thread performance**
- **Data distributions**
- **Efficient and scalable communication**
- **Load balancing and fine grain threading**
- **Parallel I/O**

Data distributions

- **Affinity directives are already in the language**
 - block-cyclic distributions: `shared [B] int A[N];`
- **Simple extensions for data distributions, for example multi-dimensional tiles**
 - `shared [B1][B2] int A[N1][N2];`
- **Allow better control of data distributions that favour particular application domains (Linear Algebra)**
- **Enable interfacing with existing high performance libraries**

UPC matrix multiplication with multidimensional tiling

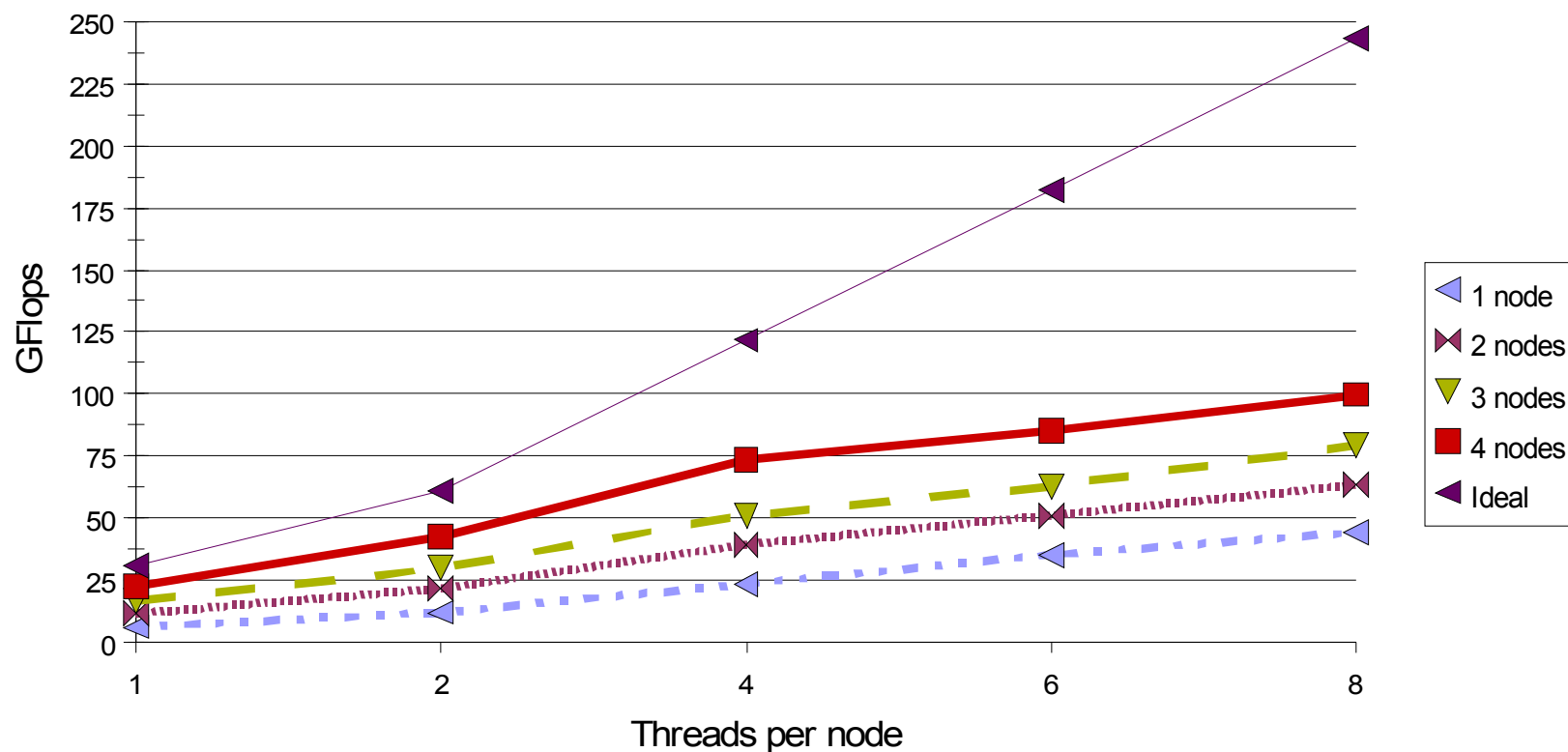
```
#define BLK_SIZE (sizeof(double)*b*b)
shared [b][b] double A[M][P], B[P][N], C[M][N];

upc_forall(int ii = 0; ii < M; ii += BS; continue)
  upc_forall(int jj = 0; 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);
  }
```

Parallel Matrix Multiply in UPC



Platform: 4-way cluster of 8-way SMP Power5

Ideal: $1.9 \text{ GHz} * 4 * \text{Threads/node} * \text{Nodes}$

Data-centric Collective Operations

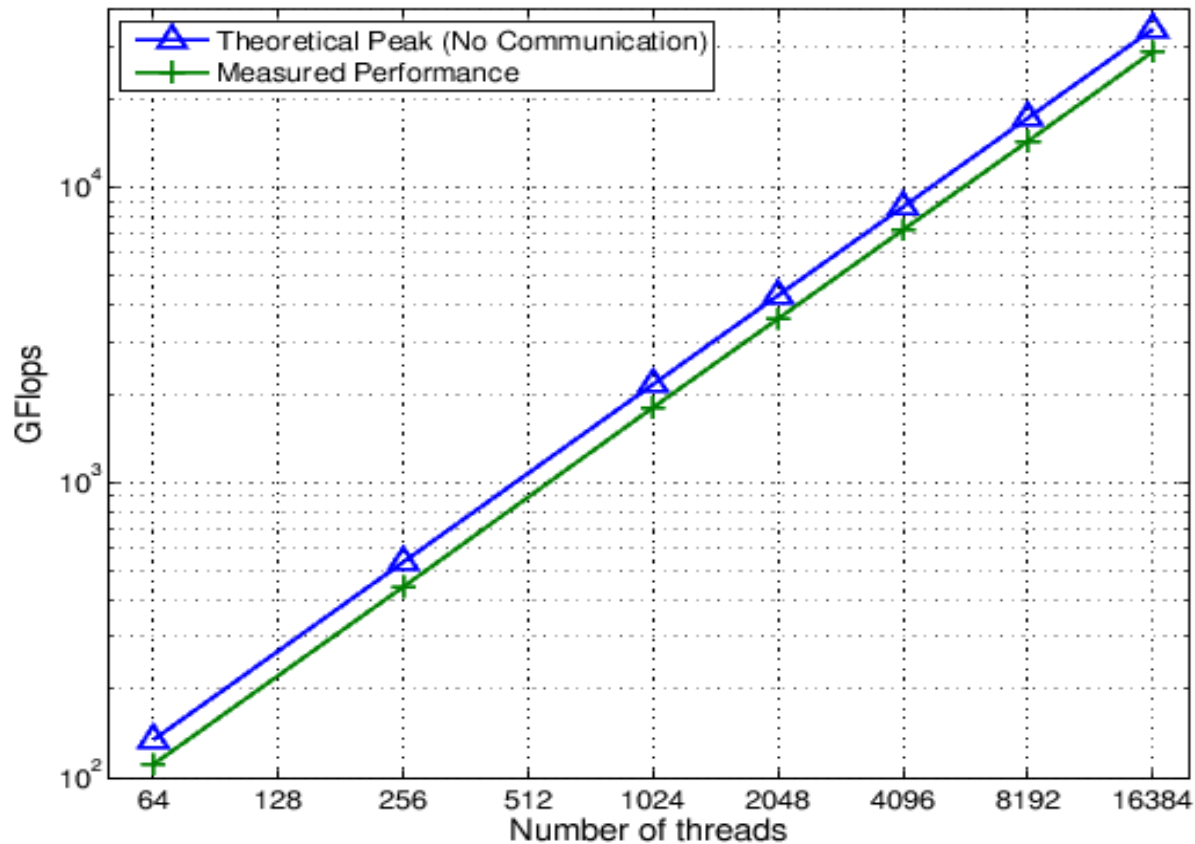
- **Current collectives are task driven – imported from MPI**
- **For a data parallel language, collectives should be expressed based on data affinity:**
 - language must be able to express data sections: Matlab like interval notation: $A\langle 0:2:N, : \rangle$
 - runtime must be able to determine participants based on affinity to data section and support teams (collectives on a subset of the tasks)
 - runtime must be able to optimize placement for efficient collective operations: user directives for processor layout

UPC matrix multiplication with collectives

```
#define BLK_SIZE (sizeof(double)*b*b)
shared [b][b] double A[M][P], B[P][N], C[M][N];
shared [b][b] double scratchA[b*Tx][b*Ty], scratchB[b*Tx][b*Ty];
int myrow=MYTHREAD/Ty;
int mycol=MYTHREAD%Ty;

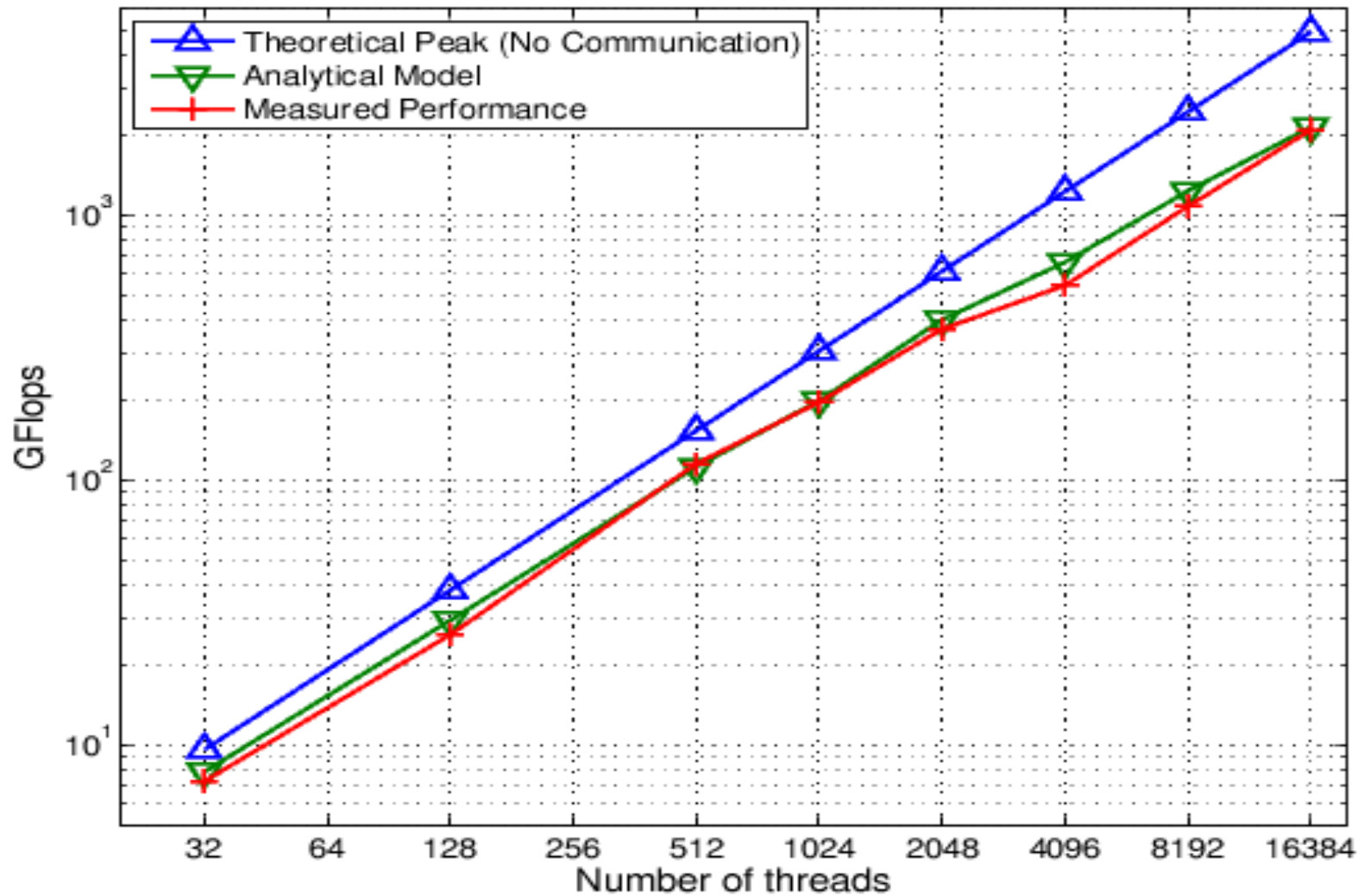
for(k=0; k<P; k+=b) {
  for(i=0; i<M; i+=Tx*b) {
    upc_stride_broadcast(scratchA<myrow, :>, &A[i+myrow*b][k], BLK_SIZE, 0);
    for(j=0; j<N; j+=Ty*b) {
      upc_stride_broadcast(scratchB<:, mycol>, &B[k][j+mycol*b], BLK_SIZE, 0);
      /* matmult*/
      dgemm((double*) &C[i+myrow*b][j+myrow*b],
            (double*) &scratchA[myrow*b][mycol*b],
            (double*) &scratchB[myrow*b][mycol*b], b, b, b);
    }
  }
}
```

Parallel Matrix Multiply in UPC



Platform: 16 racks Blue Gene/L

3D FFT Performance on Blue Gene/L



Tackling Programmer Productivity

■ **Development time**

- High level languages and language abstractions with rich semantics that allow the programmer to express algorithms and intent
- Libraries for commonly used kernels and functions

■ **Tuning and execution time**

- Compiler and runtime support that understands the semantics and exploit the underlying execution environment and architecture
- Tools that exploit architecture support for monitoring and identifying bottlenecks
- Hardware accelerators for well defined functions (computational patterns)

Summary

- **Productivity and performance are tied together**
- **Compilers are not the magic wand**
 - work well for languages and patterns that we have studied for a long time
 - have to handle general purpose languages, and therefore the number of patterns is very large
- **Programmers don't necessarily expect miracles**
 - they know the application and would be willing to express some of that knowledge – if we give them the right tools
- **Languages, libraries, runtime systems, and architectures must interact and integrate better**