Tools for HPC

APOLENT
Corporation

COREquations Division

Emerging Platforms

- Moore's Law
  - Power Wall
  - Memory Wall
  - ...
- Complex Parallel Systems are here
  - Core-to-Core comm./sync. only through L3 cache in AMD Barcelona and INTEL Nehalem proc. (L1/L2 private)
  - Processor-to-Processor comm./sync. even more costly in multi-socket configurations
  - Heterogenous multi-cores
    Cell BE, Host + GPUs (NVIDIA, AMD), Host + Larrabee

Programming Requirements

- Need to carefully examine
  - Data Partitioning
  - Communication
  - Synchronization

(A standard data-parallel implementation may be insufficient)

Relevant Applications/Domains

- Image/Video Processing, Vision, etc.
- Scientific/Engg./Statistical Computation
  - Computer Simulation of Physics, Signal Proc., SCADA/HMI
- Game Physics Engines
  - (SONY) Bullet Physics, (NVIDIA) Physx, (INTEL) Havoc
- CG Animation
  - Autodesk 3DS Max/Maya/XSI, Houdini, PIXAR PRMan, Mental Images. Also (open-source) Blender, Pixie & Aqsis
- Bioinformatics, RNA Structure Prediction
- Geospatial Information Systems (GIS)
  - ESRI ArcGIS, PCI Geomatica. Also (open-source) GRASS

Application Hotspots

- Focus of Attention (Vision, Image Processing)
  - Image filtering, edge detection, feature extraction
- Motion Estimation: H.264 (Video Processing)
- Numerical Solutions to PDEs, Filters, Regression, Control & Optimization, Parameter Estimation (Sc./Engg./Stat.)
- Dynamics (Physics Simulation)
  - Fluid/Cloth/Wire/Body Simulation in Games/Movies
- Rendering (CG Animation)
  - Global Illumination/Ray Tracing
- Dynamic Programming Kernels (Bioinformatics)
- Orthorectification (GIS)

Hotspot Characteristics

- Highly Compute-Intensive
- High Data Volume
- Very well structured kernel fragments

Possible Solutions

  - Automatically generate libraries
  - Develop highly tuned libraries

Automatic Library Generation
(The COREquations Engine)

Automatic Library Generation

- Domain Specific Compiler Engine
- Class of Problems: The Polyhedral Model
  - Incorporates most parts of four dwarfs in the ” Berkeley View* ”
    (each was expected to require an independent technology)
    - Structured Grids
    - Dense Matrix
    - Most of Dynamic Programming
    - Some of Graphical Models

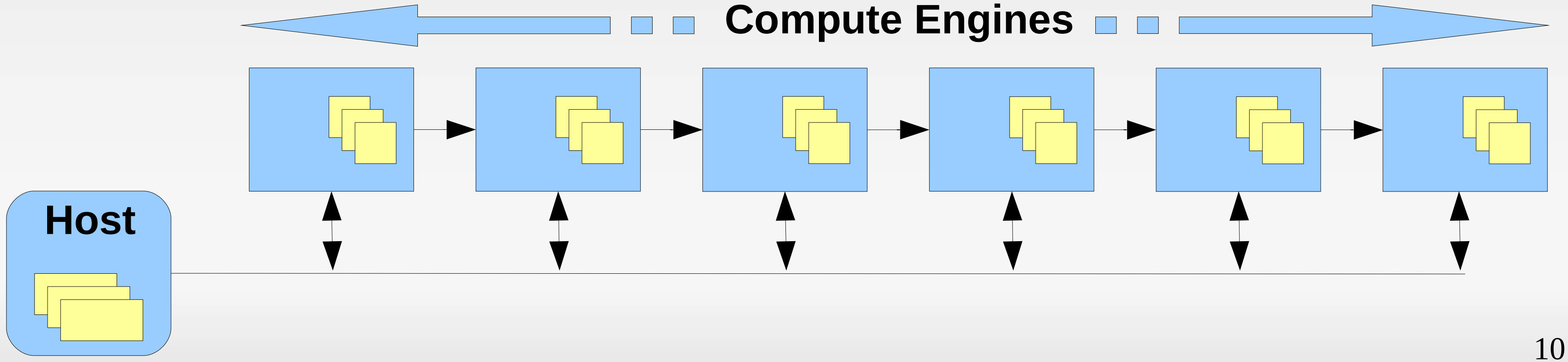
* The Berkeley View is a recent study by application/compiler/hardware specialists that outlines the most important problem domains for parallel platforms

The Polyhedral Model

- Most dynamic instr. from iterative kernels
- Manual code optimization is error prone and time consuming
- We wish to develop equivalent high level compiler optimizations
  - Analyze program behavior
  - Design custom transformations
  - Guarantee validity

Parallel Code Generation

- Host owns data (partitions, communicates & synchronizes)
- ”Compute Engines” process data
- Host reports results back to user
- Heterogeneous threads



Specifications

- Single Assignment Loops (actually equations)
- Matrix Multiplication (The HPC ”Hello World”)

```
affine matrix_product {P, Q, R|P>0 && Q>0 && R>0}
  given float A {i,k | 0<=i<P && 0<=k<Q };
        float B {k,j | 0<=k<Q && 0<=j<R };
returns float C {i,j,k | 0<=i<P && 0<=j<R && k==Q-1 };
using   float temp_C {i,j,k | 0<=i<P && 0<=j<R && -1<=k<Q };
through
temp_C[i,j,k] = case
                   { | k>=0} : temp_C[i,j,k-1]+(A[i,k]*B[k,j]);
                   { | k==-1} : 0;
                esac;
C[i,j,k] = temp_C[i,j,k];
.
```

Program Parameters

Input Variables

Output Variable

Local Variable

Equation for Accumulator

Equation for Result

Script-Driven Compilation

```
source(”Setup.bsh”);
ConnectServer(”localhost”);
ReadProgram(”matrix_product.alphabets”);
```

Read the program

```
SetMemoryMap(”matrix_product”,”C”,
        ”(P,Q,R,i,j,k->P,Q,R,i,j)”,”0,0”);
```

Desired memory layout of output

```
MPICodeGen(8,”32,32,32”);
```

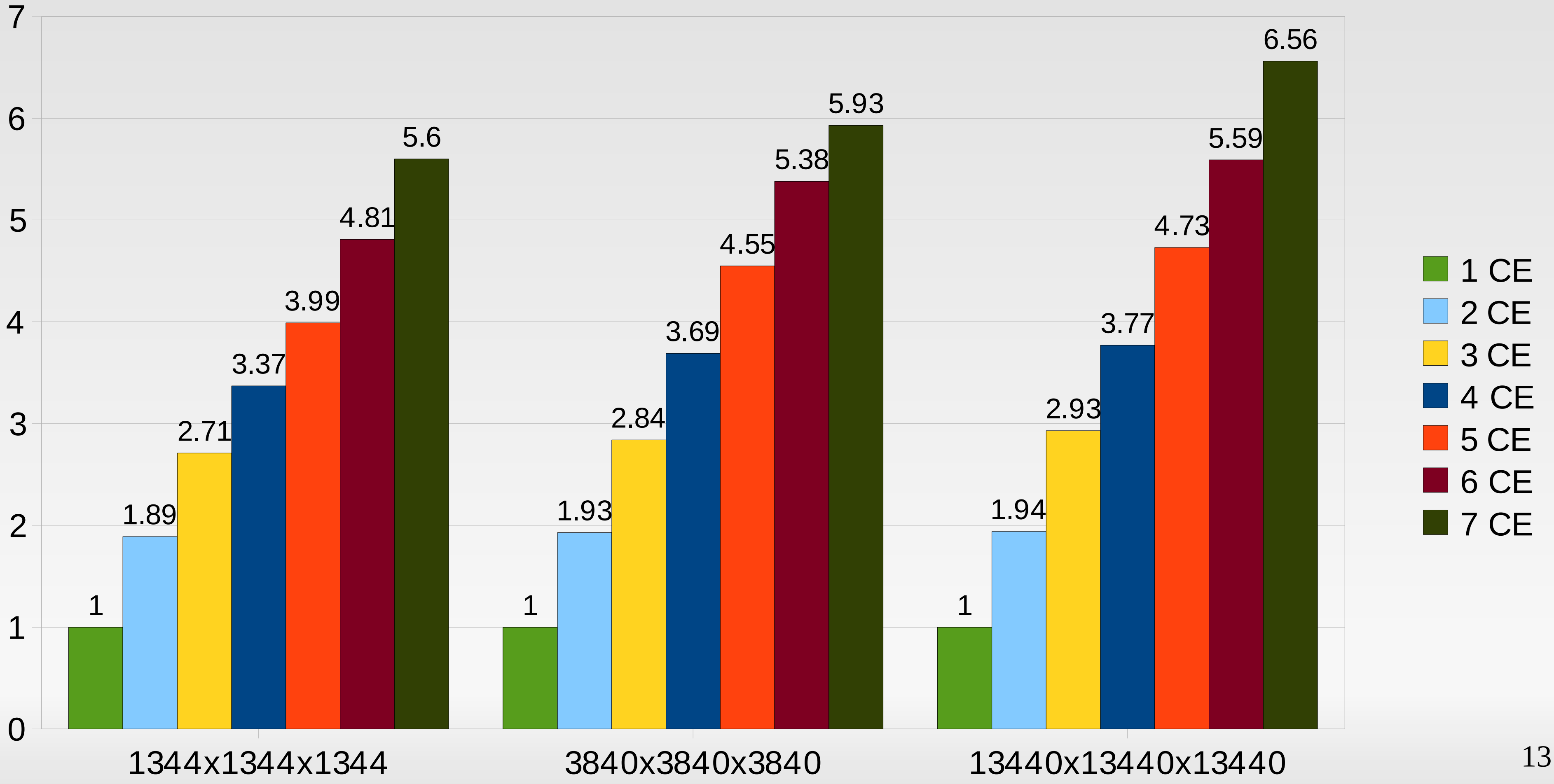
Generate code for 8 processors, tiling for parallelism with sizes 32x32x32 along i,j and k

Memory Maps are only needed for output variables

Tool finds <u>optimal memory layout</u> for local variables

Performance I: Scaling

- Near-Perfect Scaling



Performance II: Raw Comparison

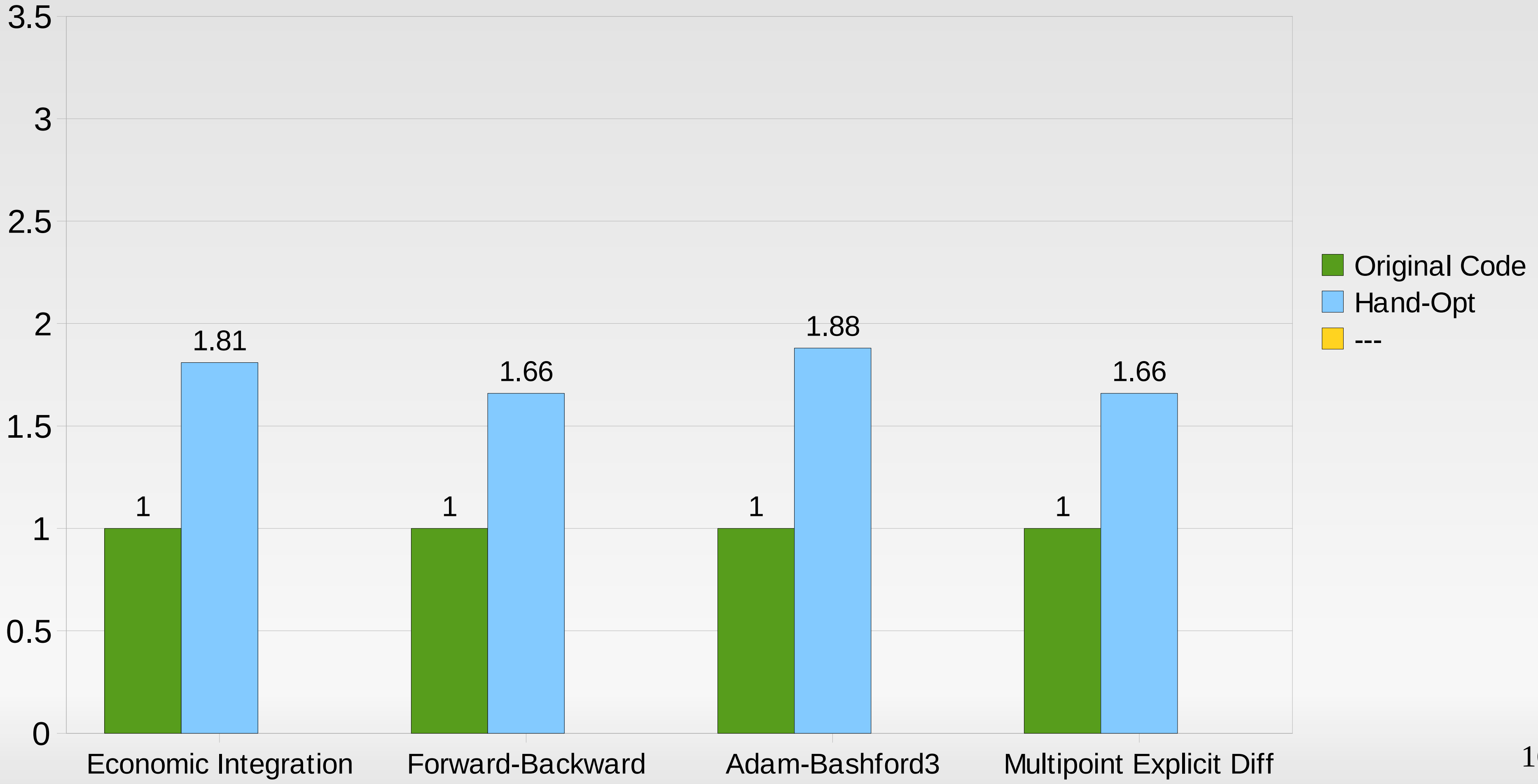
- Compare to Hand-Optimized Sequential Code
- Before, our MPI Code Generator, we wanted to generate Good Sequential Code

Sequential Code Generation

- Case Study: PDE Solver from Atmospheric Sc.
  - 2D-3D stencil. Five-point update
    - Over time, elements on an area update their value based on previous value and North, South, East and West neighbors
  - Periodic boundary conditions
    - West border depends on East border, etc.
    - SW corner depends on SE and NW corners
- Hand-Optimized by Nathan Burnett, a Masters Student at CSU
  - Performed optimizations such as loop interchange, fusion, fission, unrolling, removal of lookup-tables ...

Performance (Hand-Optimization)

- Intel CORE2 Duo (2.3Ghz)



EI PDE Solver

```
through
h[i,j,k] = case
                {| k==0}: h_init[j,i];
                {| 0<k} : h[i,j,k-1] - delt * H0 * delta[i,j,k-1];
           esac;
lap_h[i,j,k] =
 case
  {| i==0  &&  j==0}   : (h[i+1,j,k] + h[2im_h-1,j,k] + h[i,j+1,k] + h[i,2jm_h-1,k]
                         - 4*h[i,j,k]) * inv_d_sq;
  ...
  {| i==0  &&  0<=j<2jm_h}: (h[i+1,j,k] + h[2im_h-1,j,k] + h[i,j+1,k]+h[i,j-1,k]
                         - 4*h[i,j,k]) * inv_d_sq;
  ...
  {| 0<i<2im_h-1  &&  0<j<2jm_h-1}: (h[i+1,j,k] + h[i-1,j,k] + h[i,j+1,k] + h[i,j-1,k]
                                  - 4*h[i,j,k]) * inv_d_sq;
 esac;
delta[i,j,k] = case
                    {| k==0}: 0;
                    {| 0<k} : delta[i,j,k-1] - delt * g * lap_h[i,j,k];
               esac;
results[j,i] = h[i,j,TMAX];
.
```

Main Variable

Key Computation

SW Corner

West Border

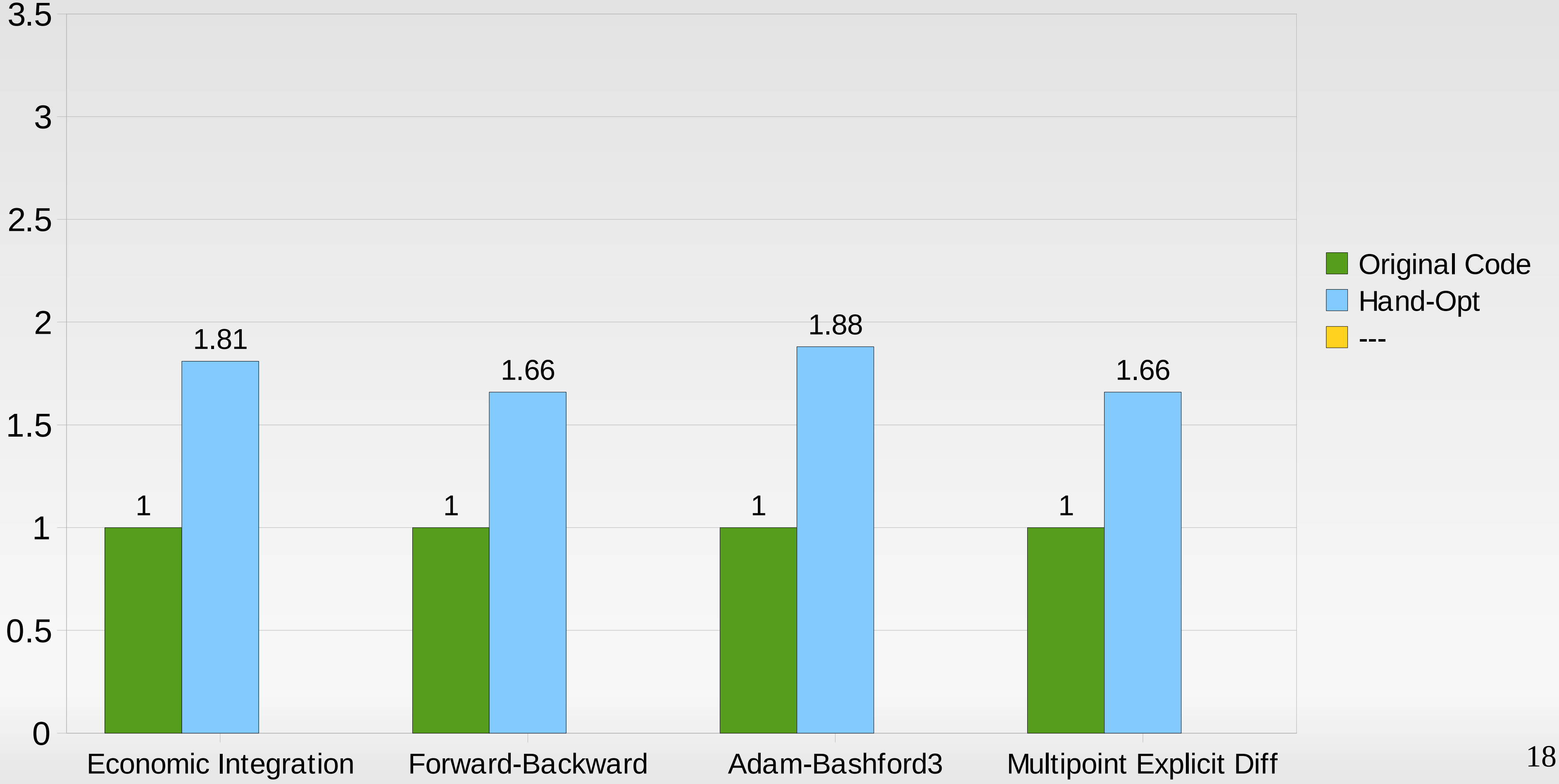
Main Body

Auxillary Variable

Output

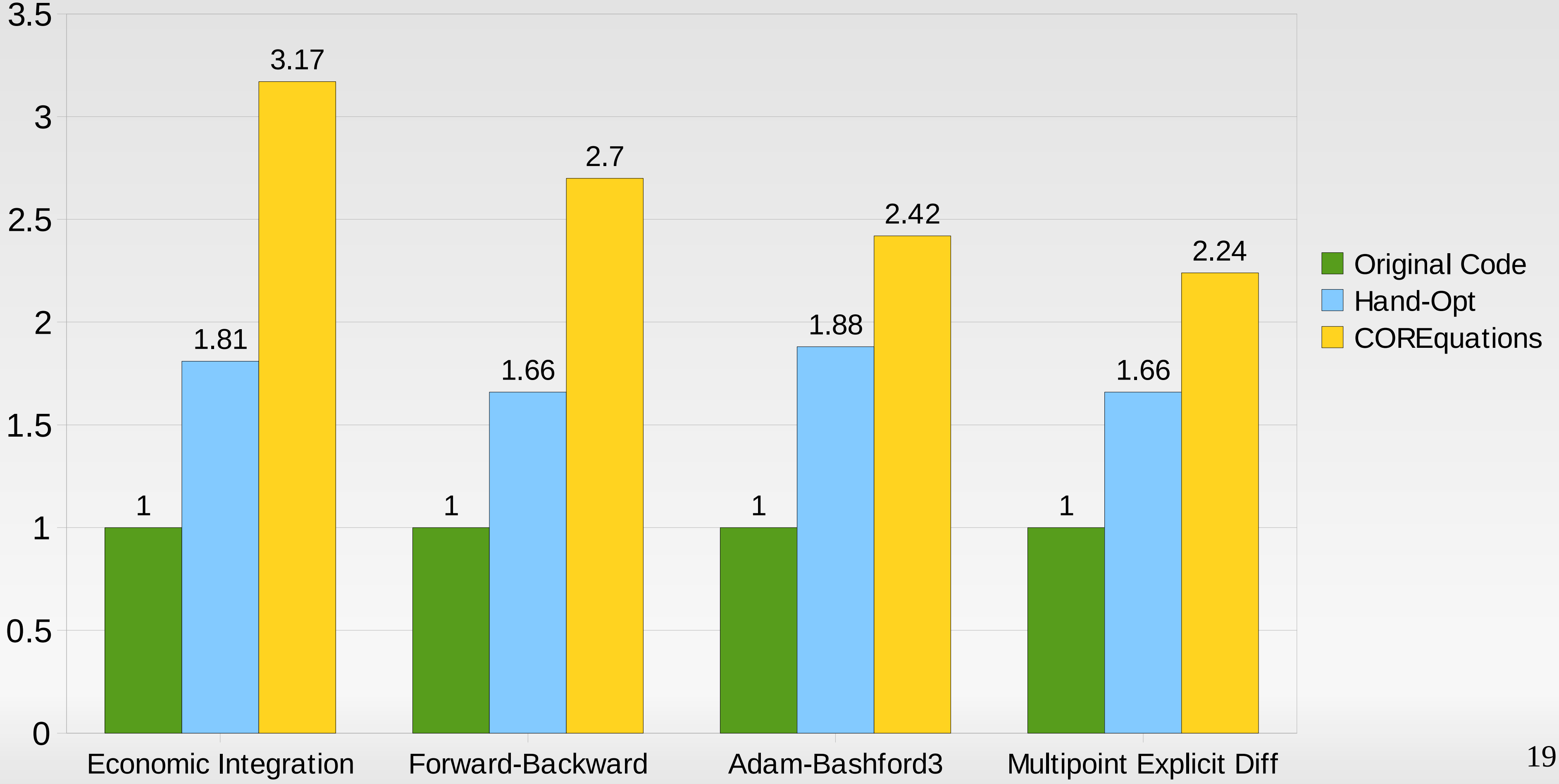
Performance (Hand-Optimization)

- Intel CORE2 Duo (2.3Ghz)



Then, I provided code for his equations

- Intel CORE2 Duo (2.3Ghz)



Why is generated code so good?

- Memory Optimization
  - (Prefetch-friendly) Alignment
  - Storage Minimization
- Loop Optimizations (just much more of them): Automatic loop-generator specializes code for <u>all</u> boundary cases
  - Aggressive Fusion
  - Aggressive Fission

Triangular Matrix Multiplication

- Beyond ”Hello World”
- P=Q=R (=N say). Add triangular constraints

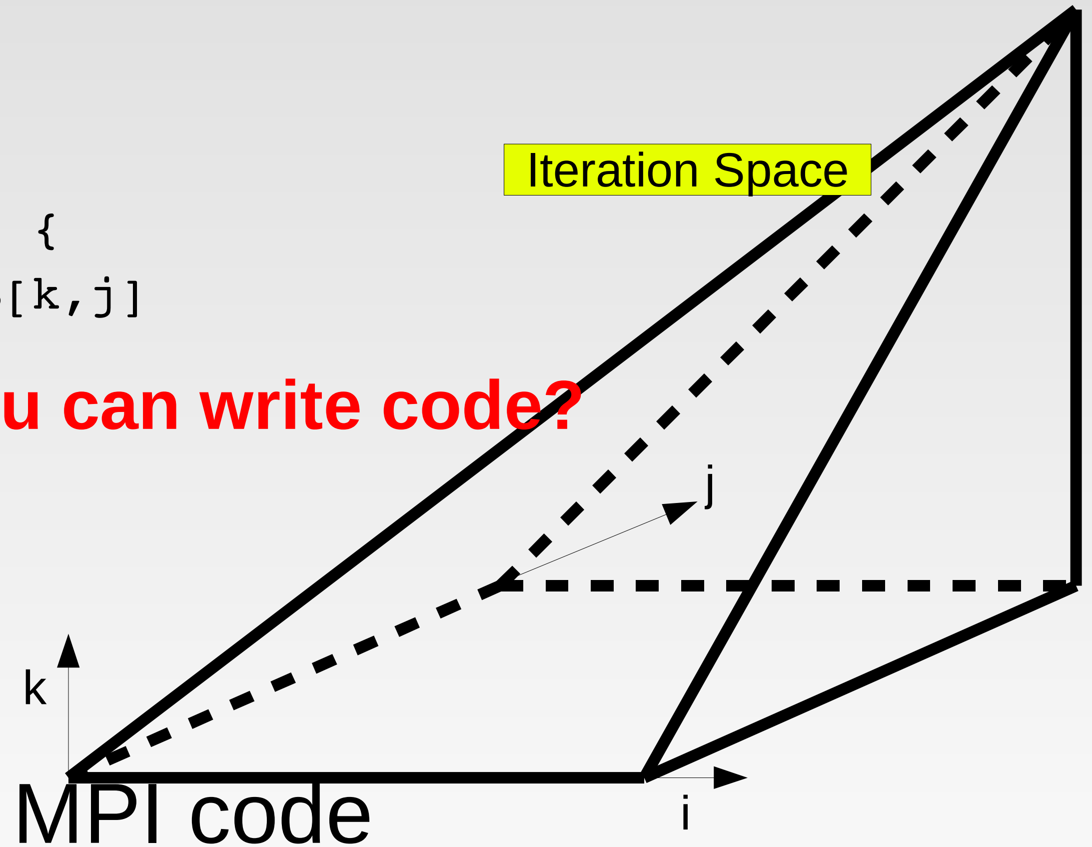
```
affine triangular_matrix_product {N|N>0}
  given float A {i,k | 0<=i<N && 0<=k<=i };
        float B {k,j | 0<=k<N && k<=j<N };
returns float C {i,j,k | 0<=i<=j && 0<=j<N && k==i } ||
                {i,j,k | 0<=i<N && 0<=j<i && k==j } ;
using   float temp_C {i,j,k | 0<=i<N && 0<=j<N && -1<=k<=(i,j) };
through
temp_C[i,j,k] = case
                   { | k>=0} : temp_C[i,j,k-1]+(A[i,k]*B[k,j]);
                   { | k==-1} : 0;
                esac;
C[i,j,k] = temp_C[i,j,k];
.
```

Triangular Matrix Multiplication

- Sequential Code

```
for i = 0 to n-1 {
    for j = 0 to n-1 {
        C[i,j] = 0;
        for k = 0 to min(i,j) {
            C[i,j] += A[i,k]*B[k,j]
        }
    }
}
```

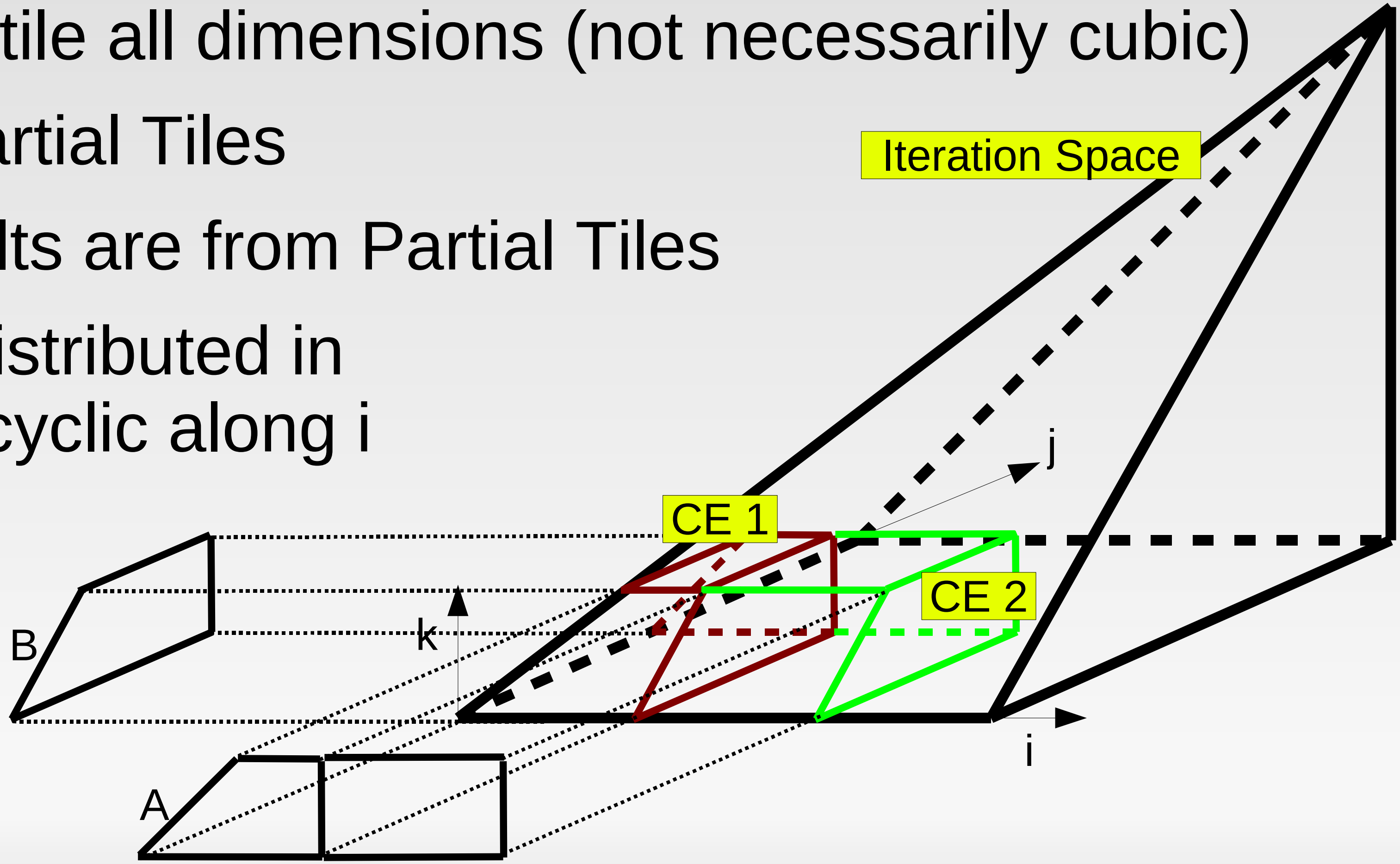
Why write equations when you can write code?



- Iteration Space is a polyhedron
- Now, let us generate MPI code (What is each Compute Engine iterating over?)

Triangular Matrix Multiplication

- Targeting memory-constrained architectures
  - Need to tile all dimensions (not necessarily cubic)
  - Many Partial Tiles
  - All Results are from Partial Tiles
  - Result distributed in
    block-cyclic along i



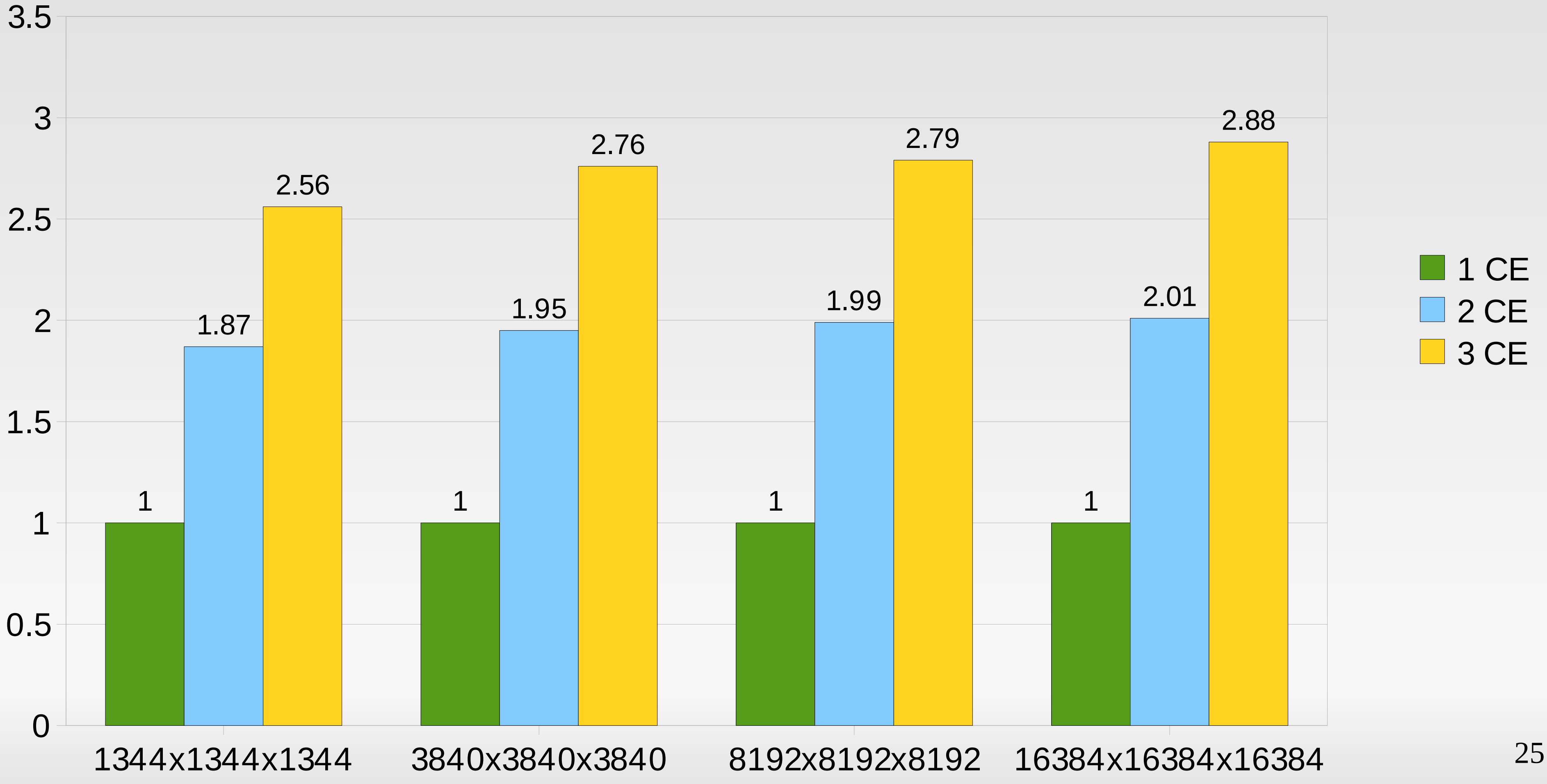
CE 1 requires a partial block of inputs from B and a partial block from A
CE 2 requires a partial block of inputs from B but a full block of inputs from A

In Short ...

- Writing MPI code even for a straightforward specialization of Matrix Multiplication is extremely time consuming and error-prone
- In COREquations, just run the same script for the new specification

Performance

- Again, Near-Perfect Scaling



Design Space Exploration

- Sequential Code for Matrix Multiplication

```
for i = 0 to n-1 {
    for j = 0 to n-1 {
        C[i,j] = 0;
        for k = 0 to N-1 {
            C[i,j] += A[i,k] * B[k,j]
        }
    }
}
```

Subsequent reads of A are along the inner index.
Memory accesses to A are prefetch friendly.

Subsequent reads of B are along the outer index.
Memory accesses to B are not prefetch friendly.

- What about writes to C?

Writes to C are to the same location throughout the inner loop

Design Space Exploration

- A number of choices in code-generation
  - Tiling: (shape and size)

    FixedTiledCodeGen(”48,32,63”);
  - Different Memory Layouts of Variables

    SetMemoryMap(”matrix_product”,”B”,
    ”(P,Q,R,k,j->P,Q,R,j,k)”,”0,0”);
  - Permutation of Loops
    - With j as the inner loop, A and C are accessed only once along the inner loop. B[k,j] becomes Prefetch-friendly.

    CoB(”matrix_product”,”C”,”(P,Q,R,i,j,k->P,Q,R,i,k,j)”);

    CoB(”matrix_product”,”temp_C”,”(P,Q,R,i,j,k->P,Q,R,i,k,j)”);

Future Work

- We're going to make it easier to program
  - Higher Level Language
    - Matrix Product should simply be specified as `C[i,j] = SUM([k], A[i,k]*B[k,j]);`
  - Domain Inferencing
- More Analysis (Scheduling, etc.)
- Get the Human-Out-of-the-Loop
- Porting to Newer Architectures (GPUs, Cell, Larrabee, ...)

Summary

- For specifications in Alphabets
  - MPI is easy
  - Design Space Exploration is easy
  - Scalable Performance
- Auto-verification
  - Compiler-assisted programming to avoid double-or incomplete-definitions, out-of-bound references ...
    - Major source of errors is eliminated

Generality of Solution

- A new language for generating HPC code for compute-intensive kernels, but also ...
  An Intermediate Representation for optimization of loops in conventional languages

Appendix

Inexpensive Design Space Exploration

Simple Matrix Product

Bsh Workspace: 0

File Font

BeanShell

2.0b4 - by Pat Niemeyer (pat@pat.net)

```
bsh % source("Setup.bsh");
bsh % ConnectServer("localhost");
bsh % ReadProgram("../../examples/embedded/matrix_product.alphabets");
bsh % SetMemoryMap("matrix_product","C","(P,Q,R,i,j,k->P,Q,R,i,j)","0,0");
bsh % CodeGen();
bsh % GenerateWrappers();
bsh %
```

Refer to script on slide 7

File Edit View Terminal Tabs Help

```
int i__ , j__ , k__ ;
#define S1(i__,k__,j__)   temp_C((i__),(k__),(j__)) = (temp_C((i__),(k__),(j__)-1))+((A((i__),(j__)))*(B((j__),(k__))))
#define S2(i__,k__,j__)   temp_C((i__),(k__),(j__)) = 0
#define S3(i__,k__,j__)   C((i__),(k__),(j__)) = temp_C((i__),(k__),(j__))
/* Generated from  by CLooG v0.14.1 gmp bits in 0.06s. */
/* CLooG asked for 17260 KBytes. */
if (Q__ >= 2) {
  for (i__=0;i__<=P__-1;i__++) {
    for (j__=0;j__<=R__-1;j__++) {
      S2(i__,j__,-1) ;
      for (k__=0;k__<=Q__-2;k__++) {
        S1(i__,j__,k__) ;
      }
      k__ = Q__-1 ;
      k__ = Q__-1 ;
      S1(i__,j__,Q__-1) ;
      S3(i__,j__,Q__-1) ;
    }
  }
}
if (Q__ == 1) {
  for (i__=0;i__<=P__-1;i__++) {
    for (j__=0;j__<=R__-1;j__++) {
      S2(i__,j__,-1) ;
      S1(i__,j__,0) ;
      S3(i__,j__,0) ;
    }
  }
}
```

Part of Generated Code

File Edit View Terminal Tabs Help

```
[ggupta@kenshin 01:31:11 temp_dir] gcc -O3 -o simple_mm matrix_product-seqC.c matrix_product-main.c
[ggupta@kenshin 01:35:05 temp_dir] ./simple_mm 1024 1024 1024
Execution time : 33.002811 sec.
[ggupta@kenshin 01:35:40 temp_dir]
```

Execution Time

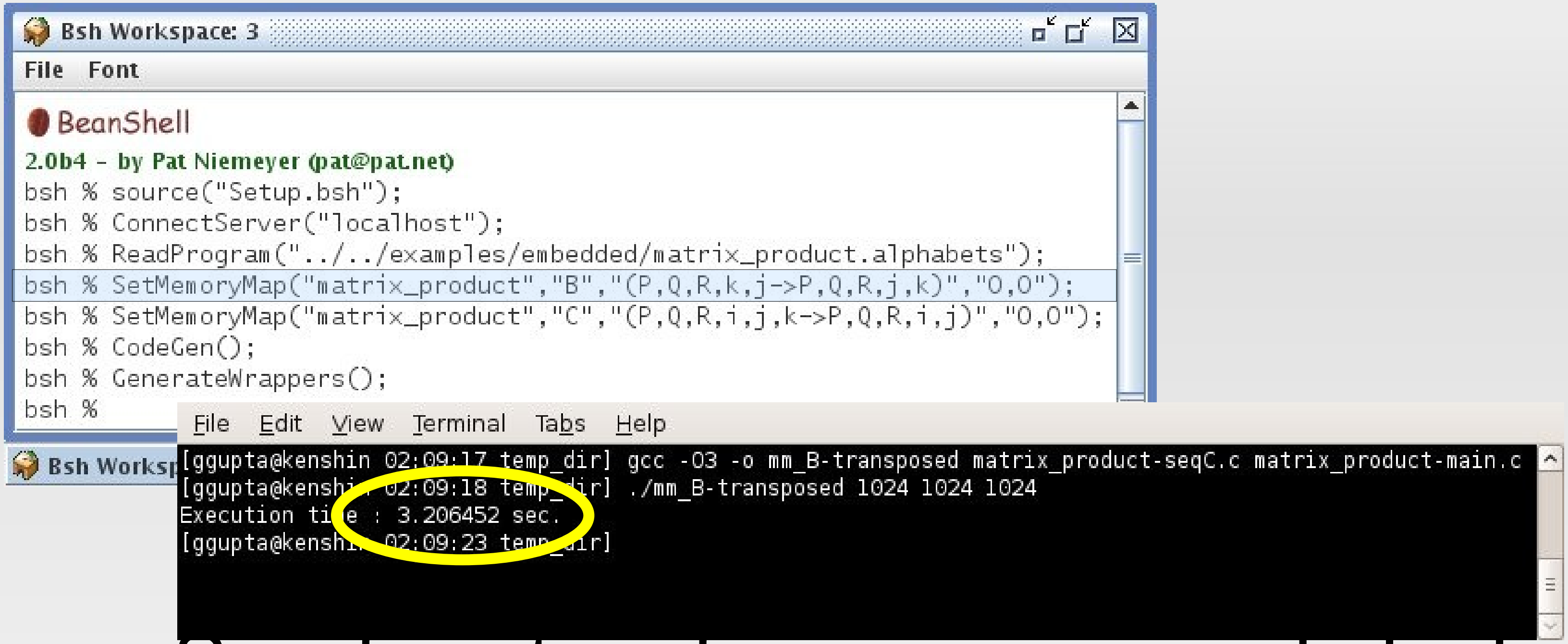
Tiled Matrix Product

```
Bsh Workspace: 2
File  Font

BeanShell
2.0b4 - by Pat Niemeyer (pat@pat.net)
bsh % source("Setup.bsh");
bsh % ConnectServer("localhost");
bsh % ReadProgram("../../examples/embedded/matrix_product.alphabets");
bsh % AShow();
affine matrix_product {P,Q,R | P>=1 && Q>=1 && R>=1}
given
   float A {i,k | k>=0 && i>=0 && Q-k>=1 && P-i>=1};
   float B {k,j | j>=0 && k>=0 && R-j>=1 && Q-k>=1};
returns
   float C {i,j,k | Q-k==1 && j>=0 && i>=0 && R-j>=1 && P-i>=1};
using
   float temp_C {i,j,k | k>=-1 && j>=0 && i>=0 && R-j>=1 && Q-k>=1 &&
P-i>=1};
through

temp_C[i,j,k] = case
   { | k>=0} : (temp_C[i,j,k-1]+(A[i,k]*B[k,j]));
   { | k==-1} : 0;
esac;

C[i,j,k] = temp_C[i,j,k];

.

bsh % SetMemoryMap("matrix_product","C","(P,Q,R,i,j,k->P,Q,R,i,j)","0,0");
bsh % FixedTiledCodeGen("32,32,32");
bsh % GenerateWrappers();
bsh %
```

Loaded Program

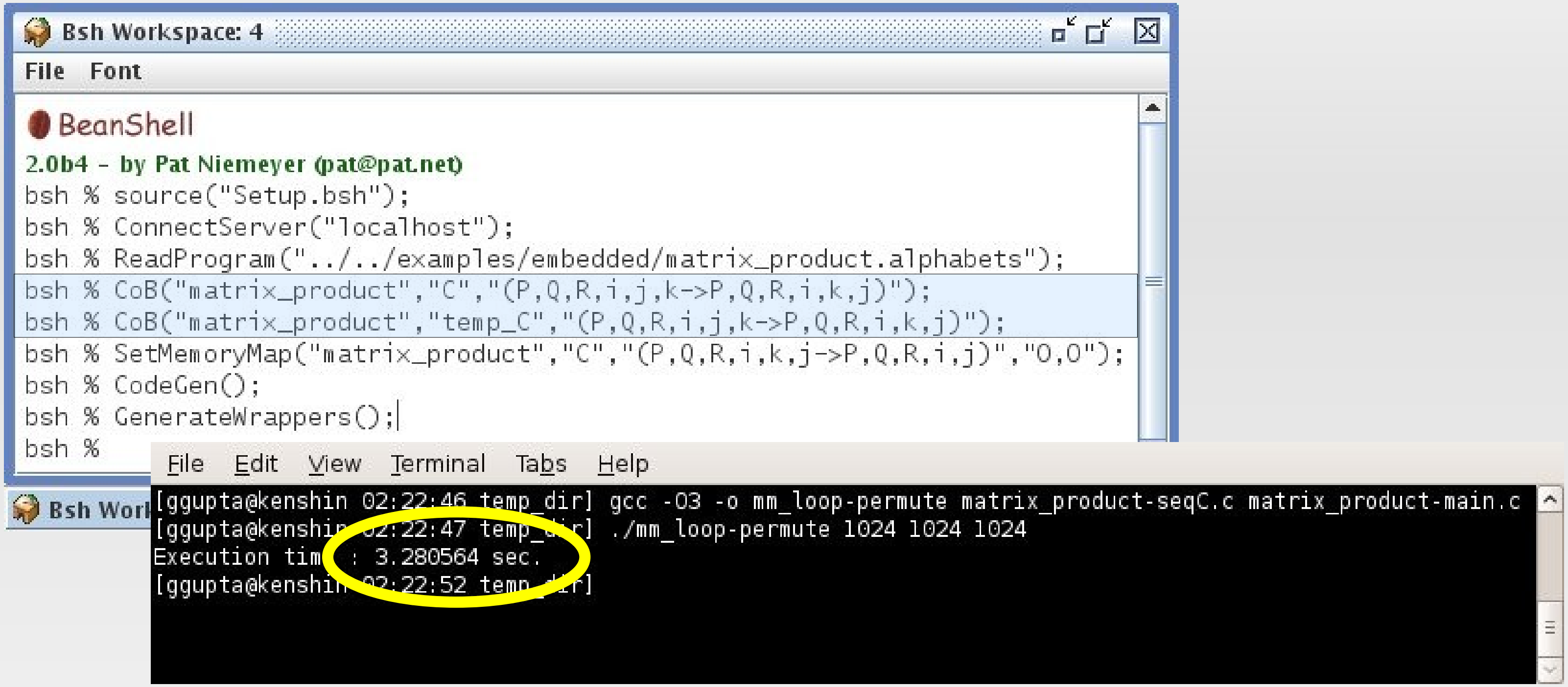
```
File  Edit  View  Terminal  Tabs  Help
[ggupta@kenshin 01:53:05 temp_dir] gcc -O3 -o tiled_mm matrix_product-fixedtiled.c matrix_product-main.c -lm
[ggupta@kenshin 01:53:08 temp_dir] ./tiled_mm 1024 1024 1024
Execution time : 15.598294 sec.
[ggupta@kenshin 01:53:25 temp_dir]
```

Changing the memory layout of B



- Significant performance upgrade by having a prefetch-friendly layout of the input array B
- Can we obtain similar performance with the standard memory layout for B

Permuting loop indices



- By having j as the innermost loop index, accesses (to arrays A and B) preserve locality