Grid Computing

2012/1/30

金光浩(11D37060 松岡研究室)

PATUS: A Code Generation and Auto-Tuning Framework For Parallel Stencil Computations

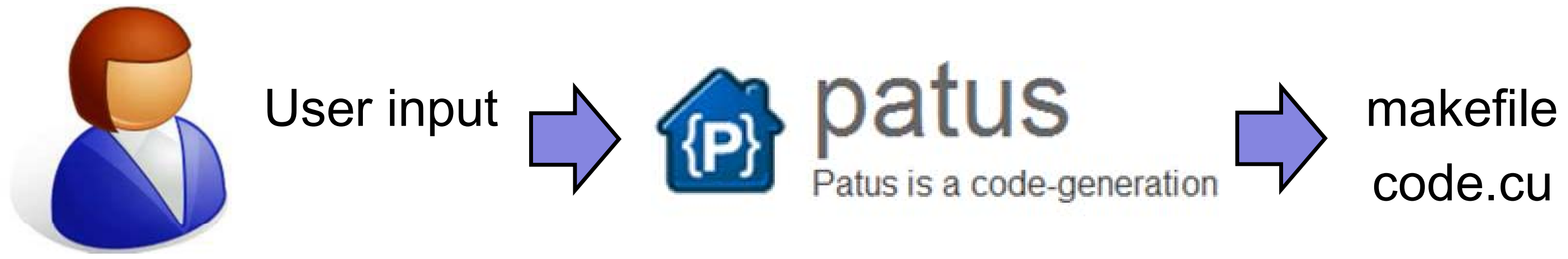
Shoaib Kamilyz, Cy Chany, Leonid Olikery,
John Shalfy, Samuel Williamsy

In Parallel & Distributed Processing Symposium (IPDPS), 2010
IEEE International

松岡研究室
金光浩　11D37060

Overview

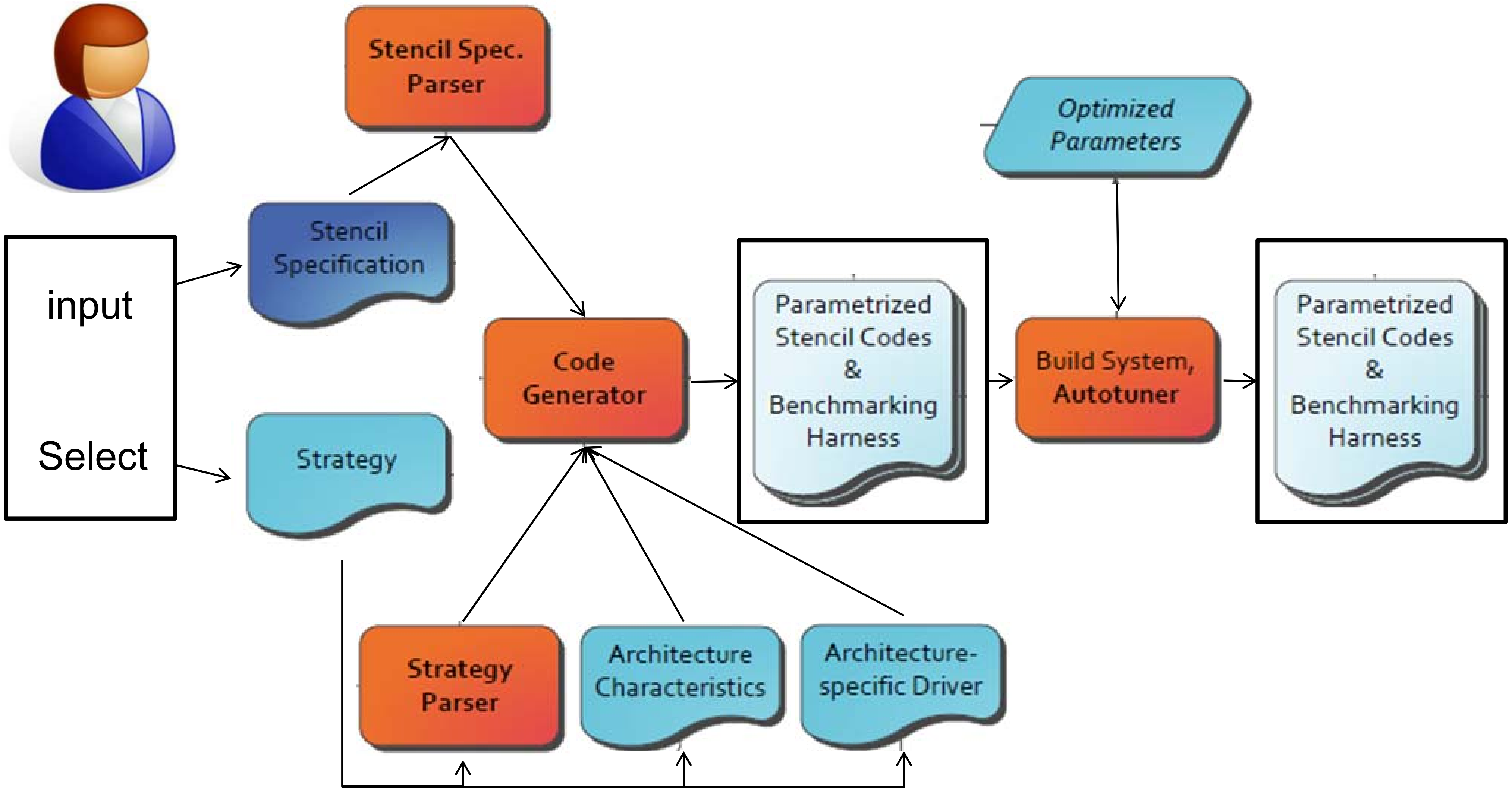
PATUS is a code generation and auto-tuning framework
for stencil computations targeted at modern multi- and
many-core processors, such as multi-core CPUs and GPUs.



Using a small domain specific language (DSL),
the user defines the stencil kernel using a C-like syntax.
The framework generates the code for a compute kernel from a specification of the stencil operation and a Strategy:

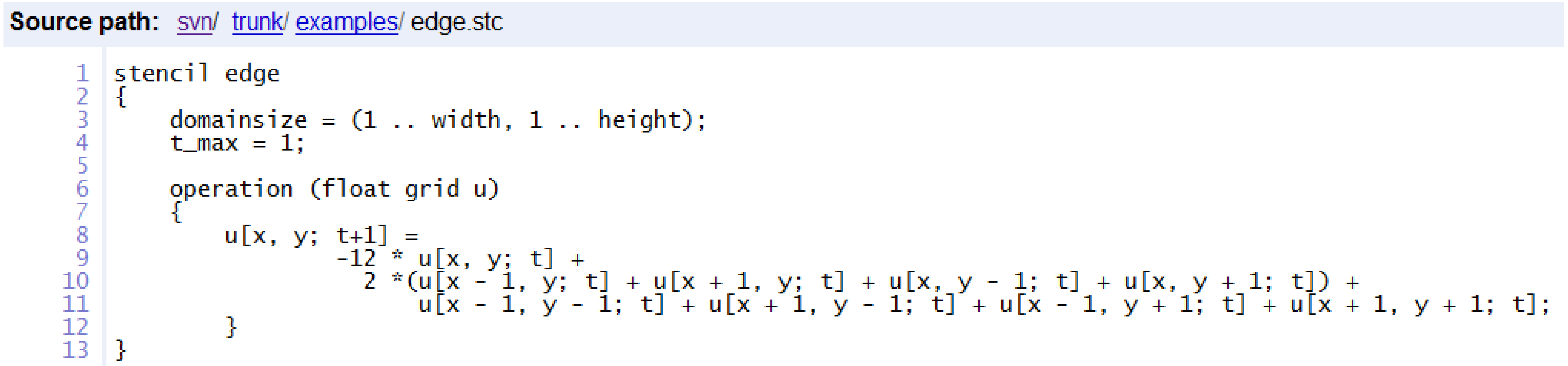
They leverage the auto-tuning methodology to find the optimal
hardware architecture-specific and Strategy-specific parameter configuration.

Process



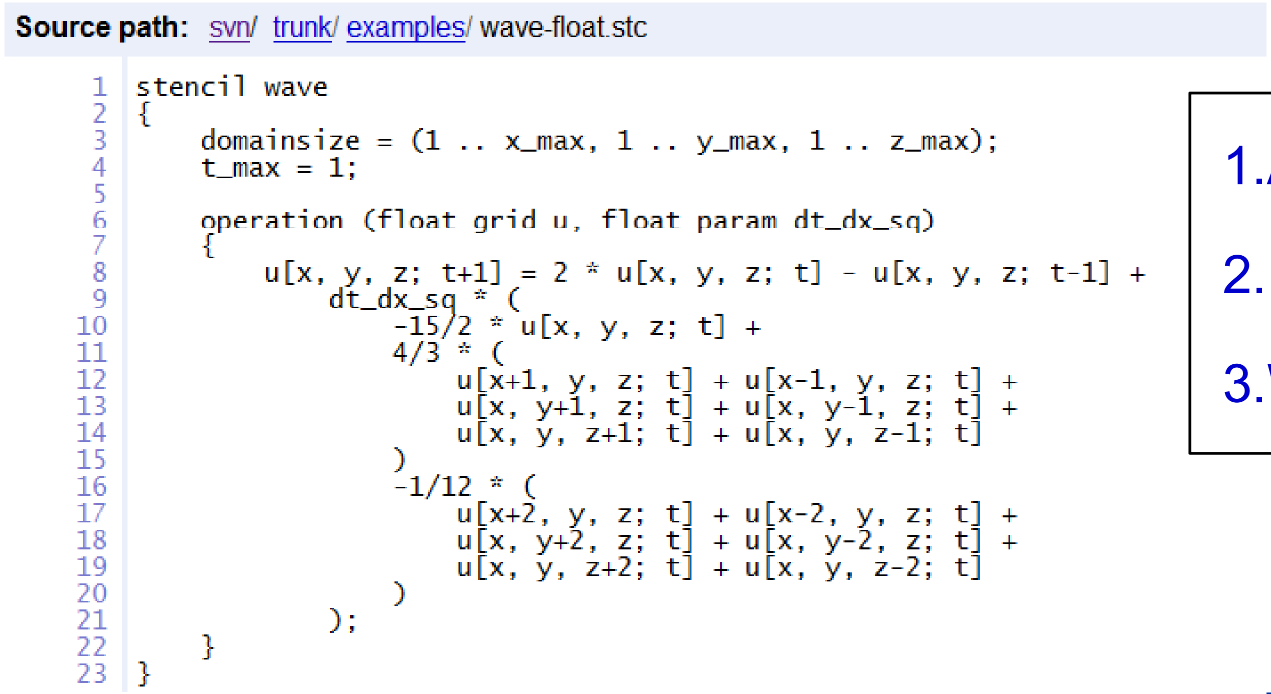
input
Select
Stencil Spec. Parser
Stencil Specification
Strategy
Code Generator
Optimized Parameters
Parametrized Stencil Codes & Benchmarking Harness
Build System, Autotuner
Parametrized Stencil Codes & Benchmarking Harness
Strategy Parser
Architecture Characteristics
Architecture-specific Driver

Examples-Input



Source path: svn/ trunk/ examples/ edge.stc

```
stencil edge
{
    domainsize = (1 .. width, 1 .. height);
    t_max = 1;

    operation (float grid u)
    {
        u[x, y; t+1] =
                -12 * u[x, y; t] +
                  2 *(u[x - 1, y; t] + u[x + 1, y; t] + u[x, y - 1; t] + u[x, y + 1; t]) +
                      u[x - 1, y - 1; t] + u[x + 1, y - 1; t] + u[x - 1, y + 1; t] + u[x + 1, y + 1; t];
        }
}
```

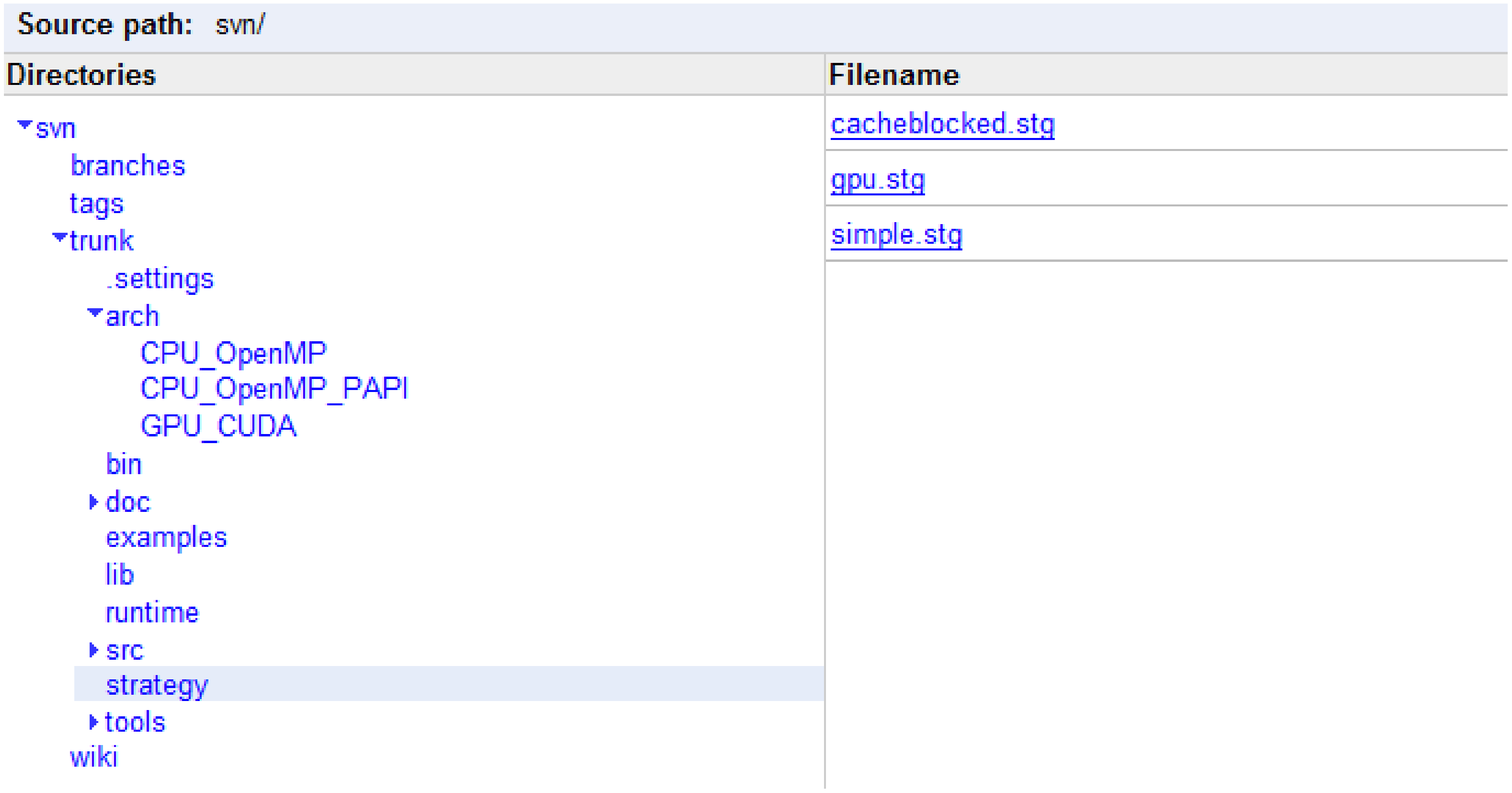


Source path: svn/ trunk/ examples/ wave-float.stc

```
stencil wave
{
    domainsize = (1 .. x_max, 1 .. y_max, 1 .. z_max);
    t_max = 1;

    operation (float grid u, float param dt_dx_sq)
    {
        u[x, y, z; t+1] = 2 * u[x, y, z; t] - u[x, y, z; t-1] +
            dt_dx_sq * (
                -15/2 * u[x, y, z; t] +
                4/3 * (
                    u[x+1, y, z; t] + u[x-1, y, z; t] +
                    u[x, y+1, z; t] + u[x, y-1, z; t] +
                    u[x, y, z+1; t] + u[x, y, z-1; t]
                )
                -1/12 * (
                    u[x+2, y, z; t] + u[x-2, y, z; t] +
                    u[x, y+2, z; t] + u[x, y-2, z; t] +
                    u[x, y, z+2; t] + u[x, y, z-2; t]
                )
            );
    }
}
```

1. A rectangular domain
2. Define Initial condition by user
3. Working on boundary condition

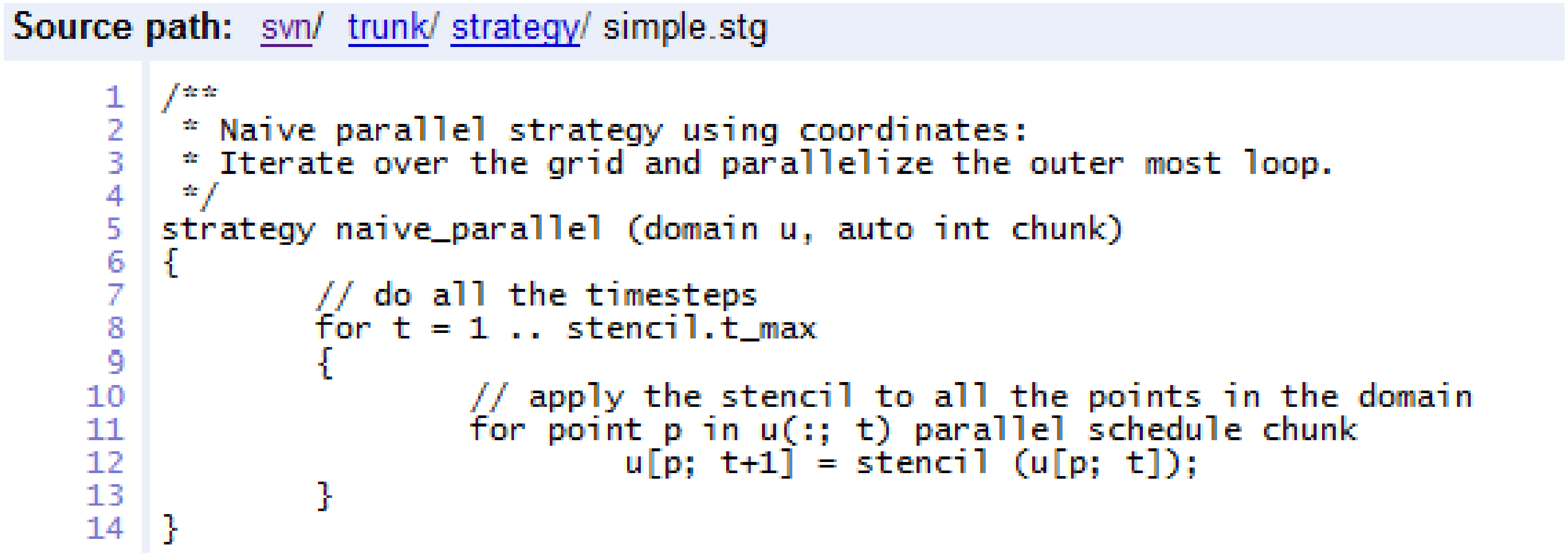
Strategy



Source path: svn/

Directories	Filename
▾svn	cacheblocked.stg
branches	gpu.stg
tags	simple.stg
▾trunk	
.settings	
▾arch	
CPU_OpenMP	
CPU_OpenMP_PAPI	
GPU_CUDA	
bin	
▸doc	
examples	
lib	
runtime	
▸src	
strategy	
▸tools	
wiki	

Strategy-1

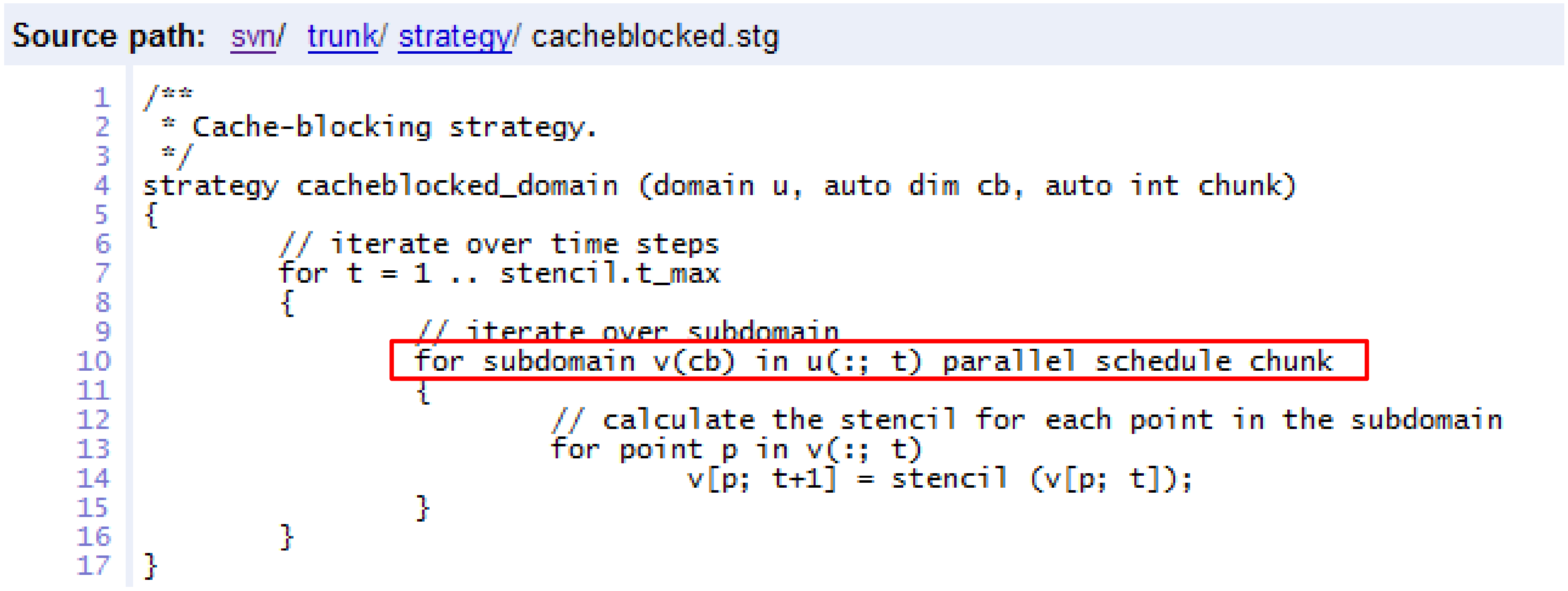


Source path: svn/ trunk/ strategy/ simple.stg

```
/**
 * Naive parallel strategy using coordinates:
 * Iterate over the grid and parallelize the outer most loop.
 */
strategy naive_parallel (domain u, auto int chunk)
{
        // do all the timesteps
        for t = 1 .. stencil.t_max
        {
                // apply the stencil to all the points in the domain
                for point p in u(:; t) parallel schedule chunk
                        u[p; t+1] = stencil (u[p; t]);
        }
}
```

The parameter chunk to the schedule keyword
defines how many consecutive blocks one thread is given.

Strategy-2



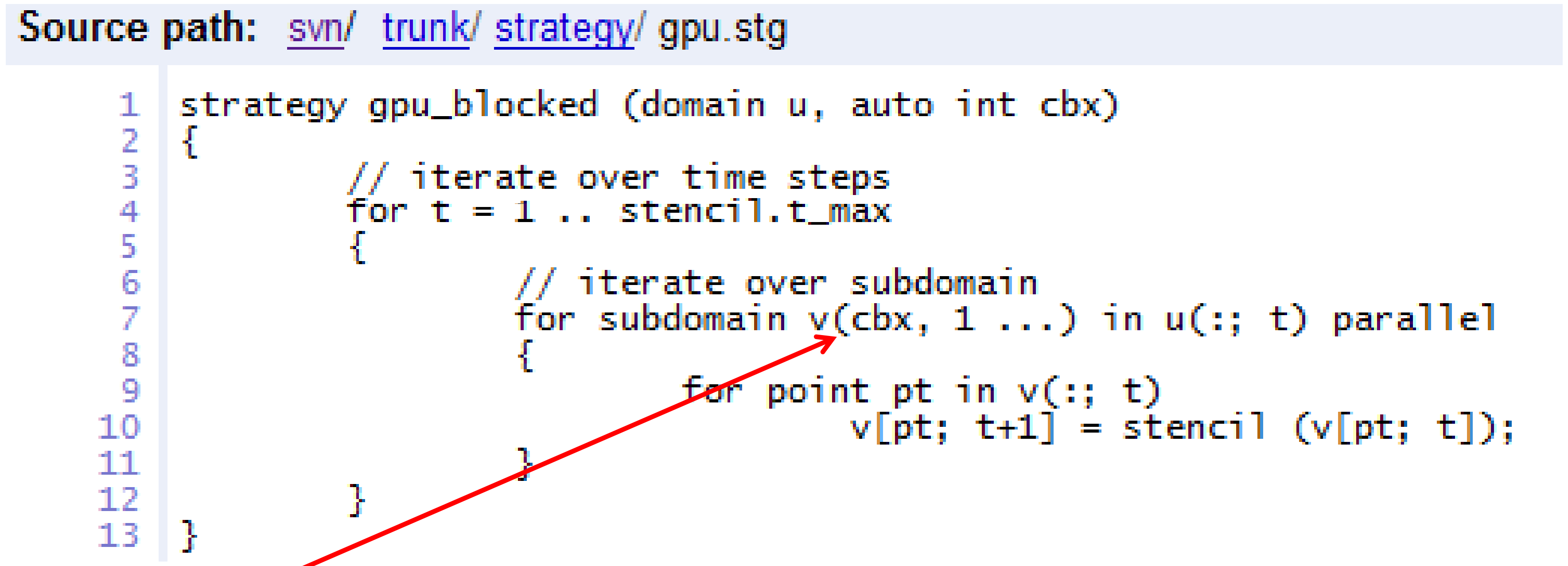
Source path: svn/ trunk/ strategy/ cacheblocked.stg

```
/**
 * Cache-blocking strategy.
 */
strategy cacheblocked_domain (domain u, auto dim cb, auto int chunk)
{
        // iterate over time steps
        for t = 1 .. stencil.t_max
        {
                // iterate over subdomain
                for subdomain v(cb) in u(:; t) parallel schedule chunk
                {
                        // calculate the stencil for each point in the subdomain
                        for point p in v(:; t)
                                v[p; t+1] = stencil (v[p; t]);
                }
        }
}
```

Define the block size[of each dimension] and access order[schedule].

blocks v of size cb over the root domain u
The cb will be interfaced with the auto tuner.
values for cb= (c1; c2; : : : ; cd), where d is the dimensionality of the stencil.

Strategy-3



Source path: svn/ trunk/ strategy/ gpu.stg

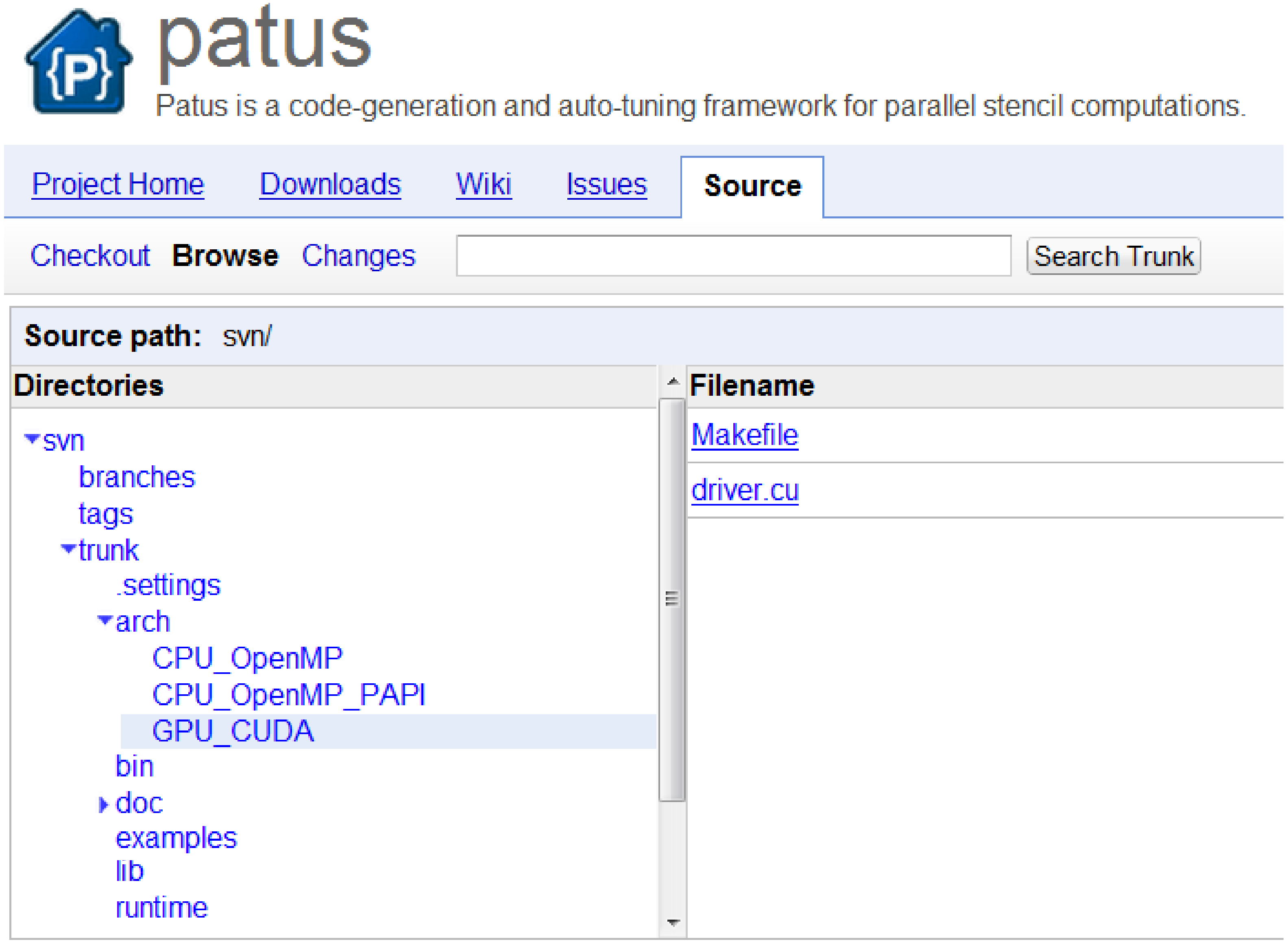
```
strategy gpu_blocked (domain u, auto int cbx)
{
        // iterate over time steps
        for t = 1 .. stencil.t_max
        {
                // iterate over subdomain
                for subdomain v(cbx, 1 ...) in u(:; t) parallel
                {
                        for point pt in v(:; t)
                                v[pt; t+1] = stencil (v[pt; t]);
                }
        }
}
```

1-dimensional thread blocks and grids,

3-dimensional thread blocks and a 2-dimensional grid,

3-dimensional thread blocks and grids

GPU-Platform



patus

Patus is a code-generation and auto-tuning framework for parallel stencil computations.

Source path: svn/

Directories

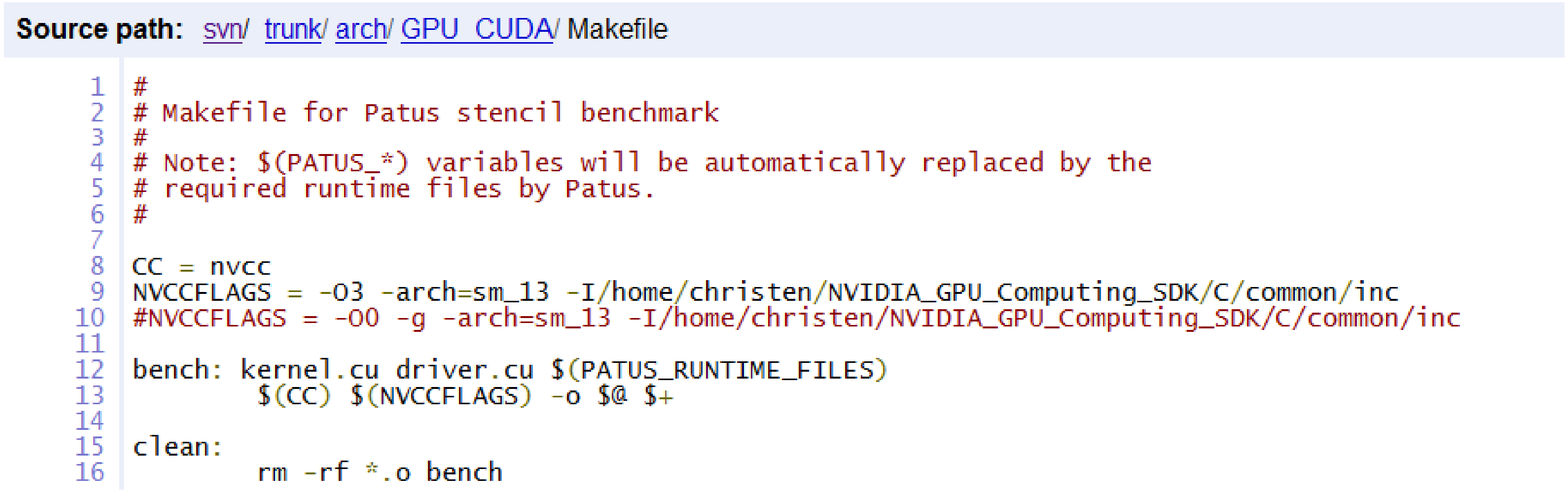
- svn
  - branches
  - tags
  - trunk
    - .settings
    - arch
      - CPU_OpenMP
      - CPU_OpenMP_PAPI
      - GPU_CUDA
    - bin
    - doc
    - examples
    - lib
    - runtime

Filename

Makefile

driver.cu

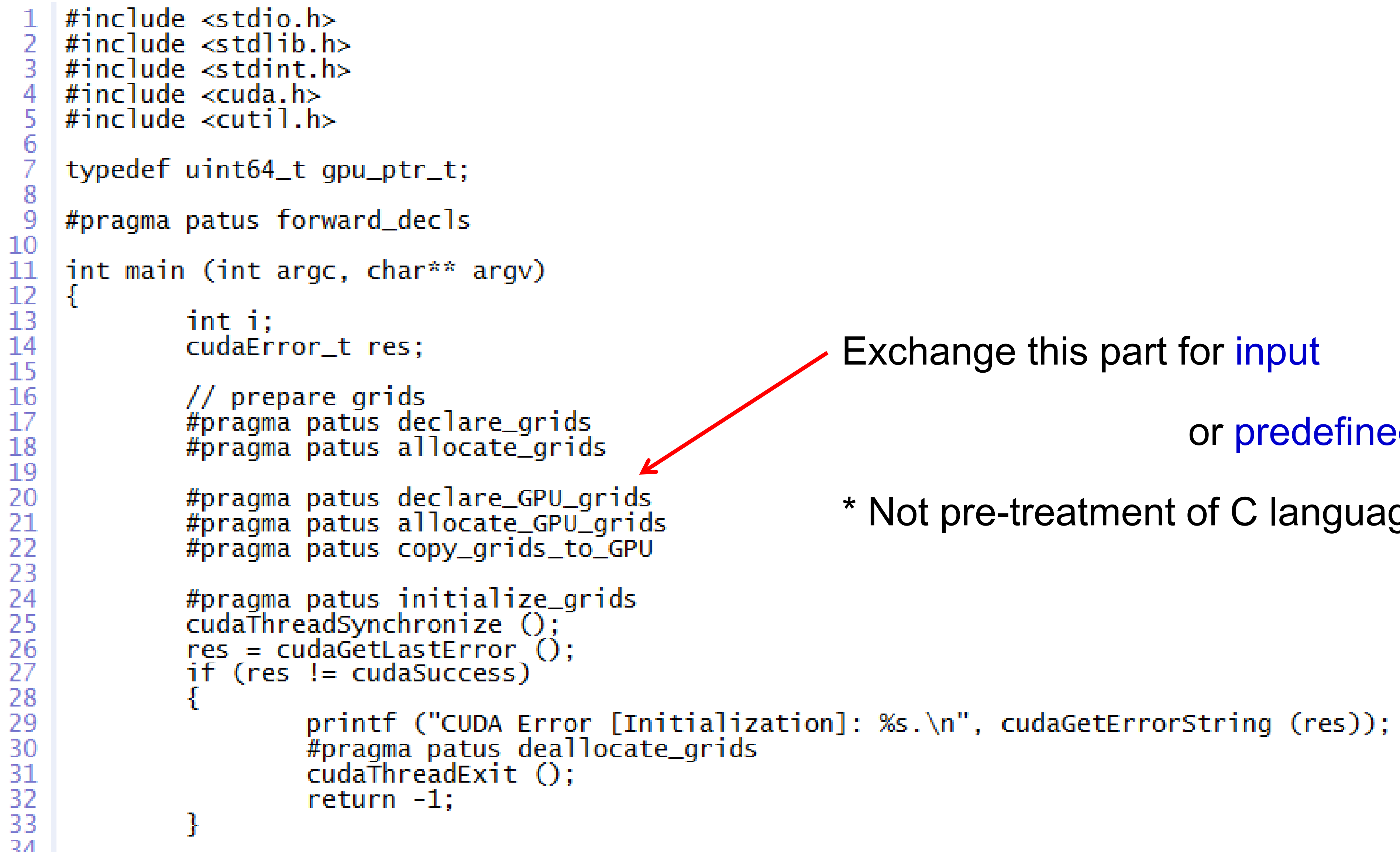
Makefile



Source path: svn/ trunk/ arch/ GPU_CUDA/ Makefile

```
#
# Makefile for Patus stencil benchmark
#
# Note: $(PATUS_*) variables will be automatically replaced by the
# required runtime files by Patus.
#

CC = nvcc
NVCCFLAGS = -O3 -arch=sm_13 -I/home/christen/NVIDIA_GPU_Computing_SDK/C/common/inc
#NVCCFLAGS = -O0 -g -arch=sm_13 -I/home/christen/NVIDIA_GPU_Computing_SDK/C/common/inc

bench: kernel.cu driver.cu $(PATUS_RUNTIME_FILES)
	$(CC) $(NVCCFLAGS) -o $@ $+

clean:
	rm -rf *.o bench
```

Driver.cu



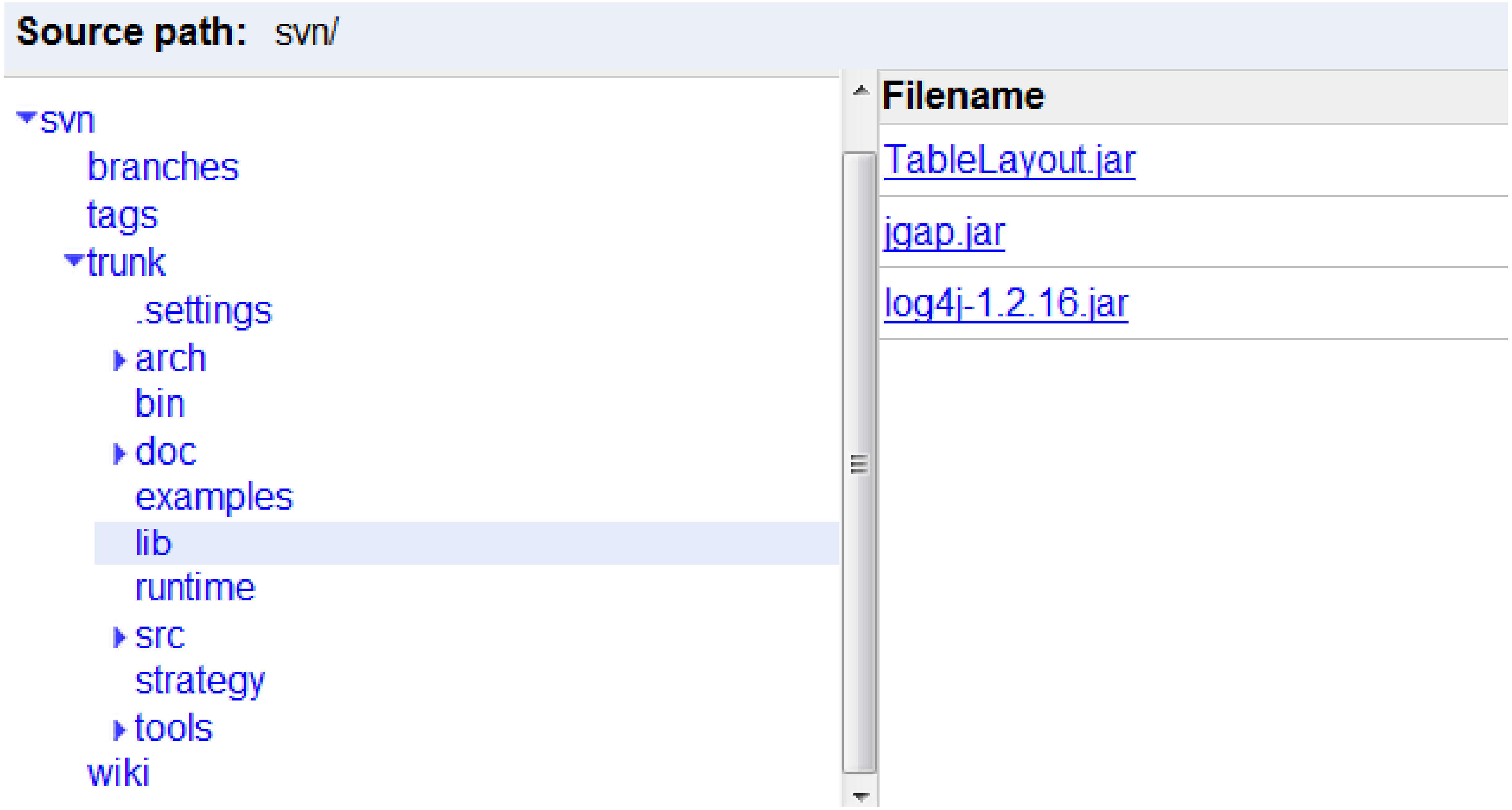
```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <cuda.h>
#include <cutil.h>

typedef uint64_t gpu_ptr_t;

#pragma patus forward_decls

int main (int argc, char** argv)
{
        int i;
        cudaError_t res;

        // prepare grids
        #pragma patus declare_grids
        #pragma patus allocate_grids

        #pragma patus declare_GPU_grids
        #pragma patus allocate_GPU_grids
        #pragma patus copy_grids_to_GPU

        #pragma patus initialize_grids
        cudaThreadSynchronize ();
        res = cudaGetLastError ();
        if (res != cudaSuccess)
        {
                printf ("CUDA Error [Initialization]: %s.\n", cudaGetErrorString (res));
                #pragma patus deallocate_grids
                cudaThreadExit ();
                return -1;
        }
```

Exchange this part for input

or predefined

* Not pre-treatment of C language

Java interface



Source path: svn/

- svn
  - branches
  - tags
  - trunk
    - .settings
    - arch
    - bin
    - doc
    - examples
    - lib
    - runtime
    - src
    - strategy
    - tools
  - wiki

Filename
TableLayout.jar
jgap.jar
log4j-1.2.16.jar

Experiment

The Anelastic Wave Propagation code AWP-ODC
of the Southern California Earthquake Center (SCEC)

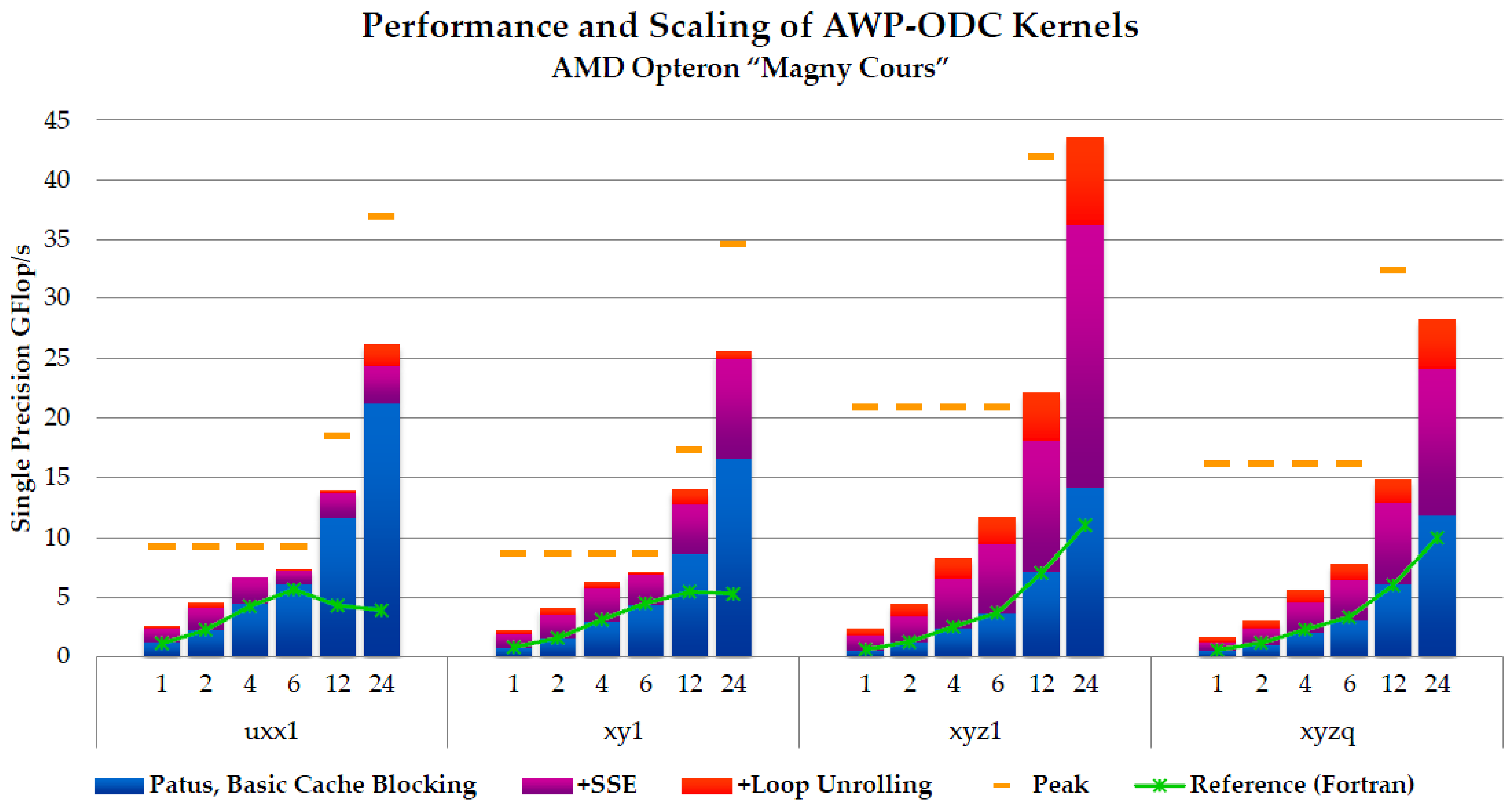
AMD Opteron “24 Cours” and an NVIDIA C2050 Fermi GPU

GNU gfortran/gcc 4.5.2 compilers
CUDA 4.0 and NVIDIA’s nvcc compiler

a 188×188×152 domain, single precision

```
u(i,j,k)=u(i,j,k)+(dth/d)*(
                           +c1*(xx(i,j,k)-xx(i-1,j,k))+
                           +c2*(xx(i+1,j,k)-xx(i-2,j,k))+
                           +c1*(xy(i,j,k)-xy(i,j-1,k))+
                           +c2*(xy(i,j+1,k)-xy(i,j-2,k))+
                           +c1*(xz(i,j,k)-xz(i,j,k-1))+
                           +c2*(xz(i,j,k+1)-xz(i,j,k-2)))
```

CPU-result



Performance and Scaling of AWP-ODC Kernels
AMD Opteron "Magny Cours"
Single Precision GFlop/s
0
5
10
15
20
25
30
35
40
45
1
2
4
6
12
24
uxx1
xy1
xyz1
xyzq
Patus, Basic Cache Blocking
+SSE
+Loop Unrolling
Peak
Reference (Fortran)

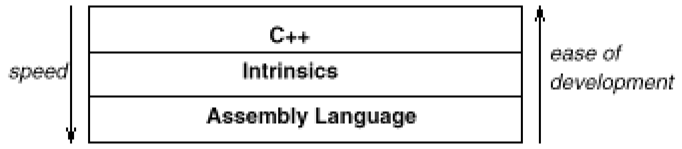
SSE

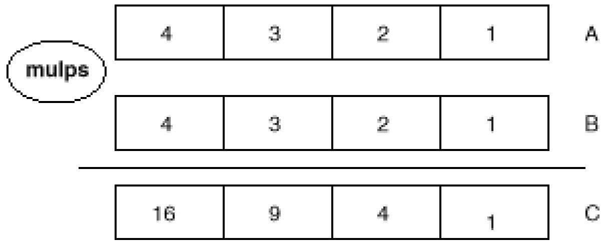
The Intel Streaming SIMD Extensions technology enhances the performance of floating-point operations. Intel C\C++, Microsoft's Macro Assembler support SSE. Pentium III and Pentium III Xeon SIMD instructions can greatly increase performance when exactly the same operations are to be performed on multiple data objects.

Intel's first IA-32 SIMD effort was the MMX instruction set. MMX had two main problems: it re-used existing floating point registers making the CPU unable to work on both floating point and SIMD data at the same time, and it only worked on integers.

SSE floating point instructions operate on a new independent register set (the XMM registers), and it adds a few integer instructions that work on MMX registers.

```
#include <xmmintrin.h>
__m128 a, b, c;
a = _mm_set_ps(4, 3, 2, 1)
b = _mm_set_ps(4, 3, 2, 1)
c = _mm_set_ps(0, 0, 0, 0)
c = _mm_mul_ps(a, b);
```





Arithmetic Instructions
- addps, addss
- subps, subss
- mulps, mulss
- divps, divss
- sqrtps, sqrtss
- maxps, maxss
- minps, minss

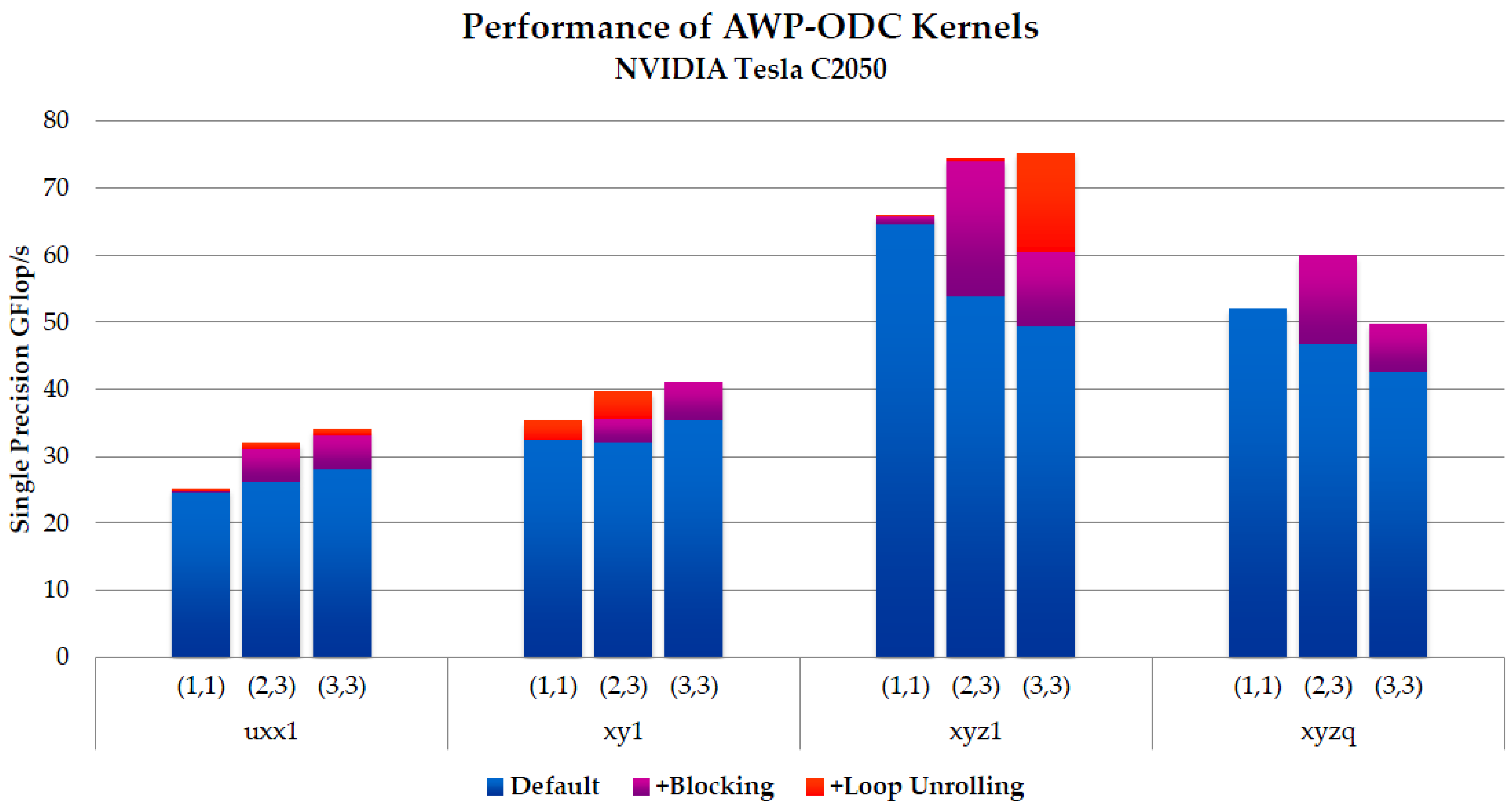
Loop unrolling

```
for (i = 0; i < 100; i++)
      A[i] = A[i] + B[i] * C
```

You can unroll the loop, as we have below, giving you the same operations in fewer iterations with less loop overhead. You can imagine how this would help on any computer. Because the computations in one iteration do not depend on the computations in other iterations, calculations from different iterations can be executed together. On a superscalar processor, portions of these four statements may actually execute in parallel:

```
for (i = 0; i < 100; i += 4)
{
         A[i]  = A[i] + B[i] * C;
     A[i+1]  = A[i+1] + B[i+1] * C;
     A[i+2]  = A[i+2] + B[i+2] * C;
     A[i+3]  = A[i+3] + B[i+3]  * C
}
```

GPU



1-dimensional thread blocks and grids,

3-dimensional thread blocks and a 2-dimensional grid,

3-dimensional thread blocks and grids

Conclusion

PATUS, a code generation and
auto-tuning framework for general stencil computations.

It is for both programmers in need of an efficient implementation of a stencil kernel for a given hardware architecture, but who do not want to care about hardware-specific tuning, and for domain experts who want to experiment.

The framework still has limitations (restriction to shared memory architectures, no special boundary treatment, lacking support for temporal blocking schemes),which we intend to overcome in the future.

PATUS is open source software and licensed under
the GNU Lesser GPL. It can be obtained from
http://code.google.com/p/patus/.

question?

Auto-tuning

Panorama [8], [9] was a research compiler for tiling iterative stencil computations in order to minimize cache misses. 2008

Berkeley stencil auto-tuner [10] seeks to substitute an annotated stencil computation in Fortran95 automatically by an optimized version. 2007

The Pochoir stencil compiler [11] applies the cache oblivious ideas initially formulated by Frigo and Strumpen [12] to stencil codes with ideally many time steps. 2008

Mint [14] targets NVIDIA GPUs as hardware platforms and translates traditional, but annotated, C code to CUDA C and applies hardware-specific optimizations specifically tailored for stencil computations.2007