Run-time code generation in C++ as a foundation for domain-specific optimization

Paul Kelly (Imperial College London)

Joint work with

Olav Beckmann, Alastair Houghton, Michael Mellor, Peter Collingbourne, Kostas Spyropoulos

Greenwich, November 200

Software Performance Optimisation Group

Imperial College London

Mission statement

- Extend optimising compiler technology to challenging contexts beyond scope of conventional compilers
- Another talk:
  - Distributed systems:
    - Across network boundaries
    - Between different security domains
    - Maintaining proper semantics in event of failures
- Another talk:
  - Active libraries for parallel scientific applications
    - Domain-specific optimisations without a DSL
- This talk:
  - Cross-component, domain-specific optimisation in numerical scientific applications, using run-time code generation

Software Performance Optimisation Group
Imperial College London

Performance programming

- Performance programming is the discipline of software engineering in its application to achieving performance goals
- This talk introduces one of the performance programming tools we have been exploring

Construction

- What is the role of constructive methods in performance programming?
- **“by construction”**
- **“by design”**
- How can we build performance into a software project?
- How can we build-in the means to detect and correct performance problems?
- As early as possible
- With minimal disruption to the software’s long-term value?

Abstraction

- Most performance improvement opportunities come from adapting components to their context
- So the art of performance programming is to figure out how to design and compose components so this doesn't happen
- Most performance improvement measures break abstraction boundaries
- This talk is about two ideas which can help:
  - Run-time program generation (and manipulation)
  - Metadata, characterising data structures, components, and their dependence relationships

Abstraction

- Most performance improvement opportunities come from adapting components to their context
- So the art of performance programming is to figure out how to design and compose components so this doesn't happen

- Most performance improvement measures break abstraction boundaries
- This talk is about two ideas which can help:
  - Run-time program generation (and manipulation)
  - Metadata, characterising data structures, components, and their dependence relationships

Abstraction

Software Performance Optimisation Group

Imperial College London

- Most performance improvement opportunities come from adapting components to their context
- So the art of performance programming is to figure out how to design and compose components so this doesn't happen
- Most performance improvement measures break abstraction boundaries
- This talk is about two ideas which can help:
  - Run-time program generation (and manipulation)
  - Metadata, characterising data structures, components, and their dependence relationships

Abstraction

- Most performance improvement opportunities come from adapting components to their context
- So the art of performance programming is to figure out how to design and compose components so this doesn't happen
- Most performance improvement measures break abstraction boundaries
- This talk is about two ideas which can help:
  - Run-time program generation (and manipulation)
  - Metadata, characterising data structures, components, and their dependence relationships

Software Performance Optimisation Group
Imperial College London

The TaskGraph library

- “**Multi-stage languages** internalize the notions of runtime program generation and execution”
  - I present a C++ library for multi-stage programming
- “**Metaprogramming** - writing programs which mess with the insides of other programs, eg those it has just generated”
  - That too!
- “**Invasive composition** - writing metaprograms to implement interesting component composition”
  - Future work

Software Performance Optimisation Group
Imperial College London

Software Performance Optimisation Group

Imperial College London

- The TaskGraph library is a portable C++ package for building and optimising code on-the-fly
- Compare:
  - `C (tcc) (Dawson Engler)
  - MetaOCaml (Walid Taha et al)
  - Jak (Batory, Lofaso, Smaragdakis)
- But there's more…

```
#include <TaskGraph>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

using namespace tg;

int main() {
  int c = 1;
  TaskGraph < Par < int, int >, Ret < int > > T;
  taskgraph( T, tuple2(x, y) ) {
    tReturn( x + y + c );
  }
  T.compile( tg::GCC );
  int a = 2;
  int b = 3;
  printf( "a+b+c = %d\n", T.execute( a, b ) );
}
```

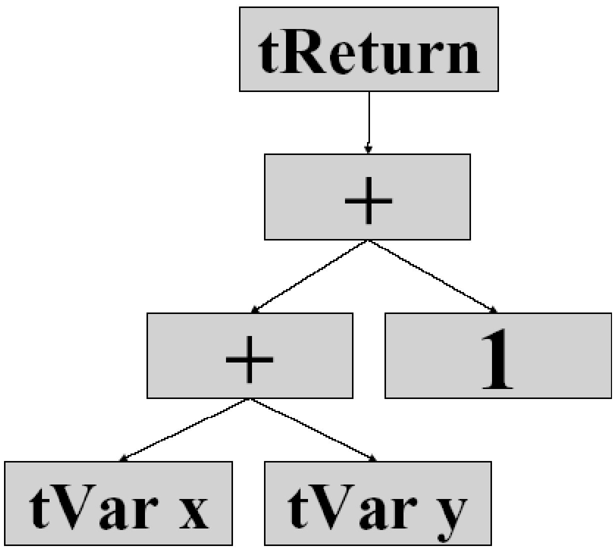
Software Performance Optimisation Group

Imperial College London

- A taskgraph is an abstract syntax tree for a piece of executable code
- Syntactic sugar makes it easy to construct
- Defines a simplified sub-language
  - With first-class multidimensional arrays, no alliasing

```
#include <TaskGraph>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
using namespace tg;
int main() {
  int c = 1;
  TaskGraph < Par < int, int >, Ret < int > > T;
  taskgraph( T, tuple2(x, y) ) {
    tReturn( x + y + c );
  }
  T.compile( tg::GCC );
  int a = 2;
  int b = 3;
  printf( "a+b+c = %d\n", T.execute( a, b ) );
}
```

- Binding time is determined by types
- In this example
  - c is static
  - x and y dynamic



- built using value of c at construction time

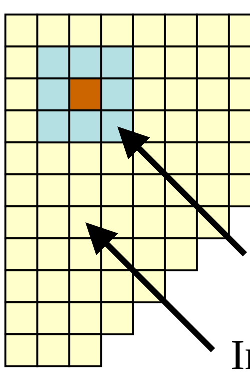
```
#include <TaskGraph>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

using namespace tg;

int main() {
  int c = 1;
  TaskGraph < Par < int, int >, Ret < int > > T;
  taskgraph( T, tuple2(x, y) ) {
    tReturn( x + y + c );
  }
  T.compile( tg::GCC );
  int a = 2;
  int b = 3;
  printf( "a+b+c = %d\n", T.execute( a, b ) );
}
```

Software Performance Optimisation Group

Imperial College London

Better example:

- Applying a convolution filter to a 2D image
- Each pixel is averaged with neighbouring pixels weighted by a stencil matrix




```
void filter (float *mask, unsigned n, unsigned m,
                const float *input, float *output,
                unsigned p, unsigned q)
{
  unsigned i, j;
  int      k, l;
  float    sum;
  int half_n = (n/2);
  int half_m = (m/2);

  for (i = half_n; i < p - half_n; i++) {
    for (j = half_m; j < q - half_m; j++) {
      sum = 0;
           // Loop bounds unknown at compile-time
           // Trip count 3, does not fill vector registers
      for (k = -half_n; k <= half_n; k++)
        for (l = -half_m; l <= half_m; l++)
          sum += input[(i + k) * q + (j + l)]
                 * mask[k * n + l];

      output[i * q + j] = sum;
    }
  }
}
```

- TaskGraph representation of this loop nest
- Inner loops are static – executed at construction time
- Outer loops are dynamic
- Uses of mask array are entirely static
- This is deduced from the types of mask, k, m and l.

emacs@SECONDSELF

Buffers Files Tools Edit Search Mule C++ Help

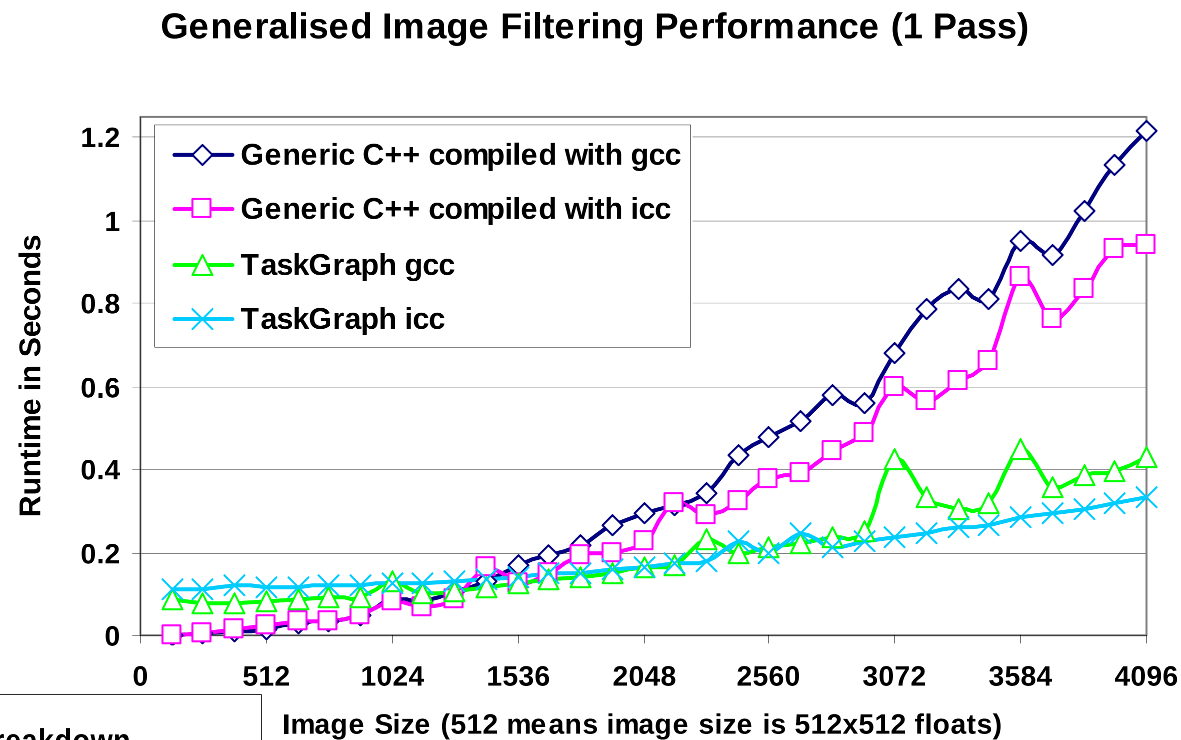
```
void specialize_convolution(
  TaskGraph < Par <float[IMG_SIZE][IMG_SIZE],
                   float[IMG_SIZE][IMG_SIZE]>,
             Ret < void > > &T,
  const int IMGSZ, const int CSZ, const float *mask )
{
  int ci, cj;
  assert( CSZ % 2 == 1 );
  const int c_half = ( CSZ / 2 );
  taskgraph( T, tuple2(tgimg, new_tgimg) ) {
    tVar ( int, i );
    tVar ( int, j );
    // Loop iterating over image
    tFor( i, c_half, IMGSZ - (c_half + 1) ) {
      tFor( j, c_half, IMGSZ - (c_half + 1) ) {
        new_tgimg[i][j] = 0.0;
        // Loop to apply convolution mask
        for( ci = -c_half; ci <= c_half; ++ci ) {
          for( cj = -c_half; cj <= c_half; ++cj) {
            new_tgimg[i][j] +=
              tgimg[i+ci][j+cj] * mask[c_half+ci][c_half+cj];
          } } } }
  }
}
```

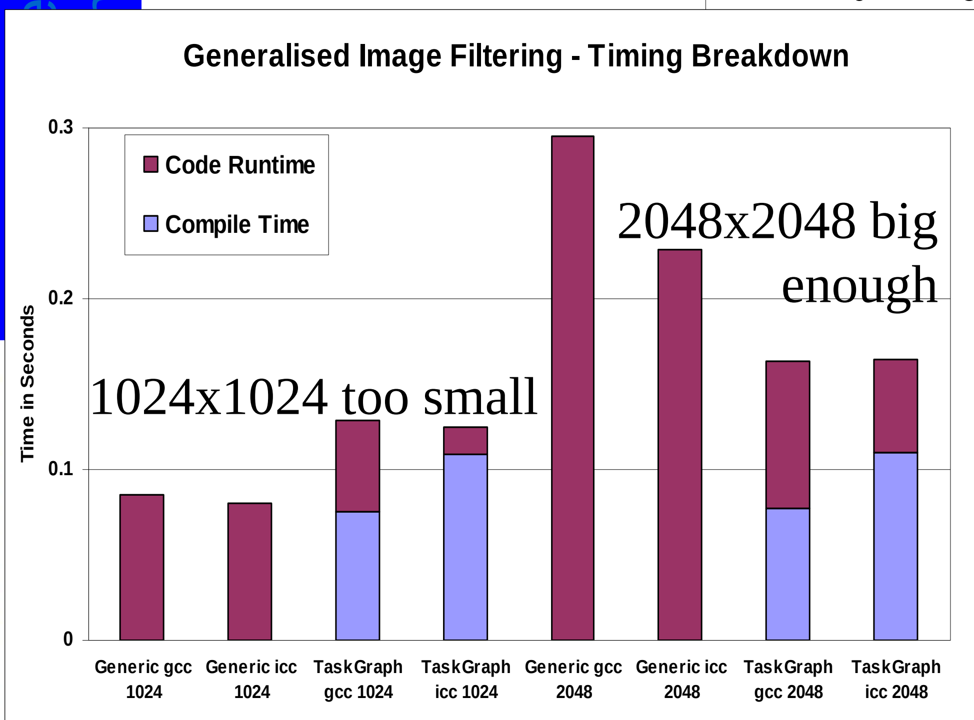
// Inner loops fully unrolled

// j loop is now vectorisable

--\-- Convolution.cc   (C++)--L4--All------------------

Image convolution using TaskGraphs: performance





- We use a 3x3 averaging filter as convolution matrix
- Images are square arrays of single-precision floats ranging in size up to 4096x4096
- Measurements taken on a 1.8GHz Pentium 4-M running Linux 2.4.17, using gcc 2.95.3 and icc 7.0
- Measurements were taken for one pass over the image

(Used an earlier release of the TaskGraph library)

Domain-specific optimisation

- The TaskGraph library is a tool for dynamic code generation and optimisation
- Large performance benefits can be gained from specialisation alone

But there's more:

- TaskGraph library builds SUIF intermediate representation
- Provides access to SUIF analysis and transformation passes
  - SUIF (Stanford University Intermediate Form)
  - Detect and characterise dependences between statements in loop nests
  - Restructure – tiling, loop fusion, skewing, parallelisation etc

Tiling

Example: matrix multiply

```
emacs@SECONDSELF
Buffers Files Tools Edit Search Mule C++ Help
typedef float MatrixType[MATRIXSIZE][MATRIXSIZE];
typedef TaskGraph< Par<MatrixType, MatrixType, MatrixType>,
                   Ret<void> > mm_TaskGraph;
float MatrixType a, b, c;

void taskMatrixMult (
  mm_TaskGraph &t,
  TaskLoopIdentifier *loop )
{
  taskgraph ( t, tuple3(a, b, c) ) {
    tVar ( int, x );
    tVar ( int, y );
    tVar ( int, z );

    tGetId ( loop[0] ); // label
    tFor ( x, 0, MATRIXSIZE - 1 ) {
      tGetId ( loop[1] ); // label
      tFor ( z, 0, MATRIXSIZE - 1 ) {
        tGetId ( loop[2] ); // label
        tFor ( y, 0, MATRIXSIZE - 1 ) {
          c[x][y] += a[x][z] * b[z][y];
        }}}}}
--\--  MM.cc          (C++)--L5--Top--
```

Original TaskGraph for matrix multiply

```
emacs@SECONDSELF
Buffers Files Tools Edit Search Mule C++ Help
main () {
  int bestTime; int bestSize = 0;
  for (int tsz = 4; tsz <= MATRIXSIZE; ++tsz) {
    int trip3 = { tsz, tsz, tsz };
    TaskLoopIdentifier loop[3];
    mm_TaskGraph MM;

    taskMatrixMult(loop, MM);
    interchangeLoops(loop[1], loop[2]);
    tileLoop(3, &loop[0], trip3);

    MM.compile(TaskGraph::ICC);

    tt3 = time_function();
    MM.execute(A, B, C);
    time = time_function()-tt3;

    if (time < bestTime || bestSize == 0) {
      bestTime = time; bestSize = tsz;
    }}}
--\--  MM.cc          (C++)--L38--Bot--------
```

Loop tries all tile sizes and finds fastest

Loop interchange and tiling

Buffers Files Tools Edit Search Mule C++ Help

```
typedef float MatrixType[MATRIXSIZE][MATRIXSIZE];
typedef TaskGraph< Par<MatrixType, MatrixType, MatrixType>,
                   Ret<void> > mm_TaskGraph;
float MatrixType a, b, c;

void taskMatrixMult (
  mm_TaskGraph &t,
  TaskLoopIdentifier *loop )
{
  taskgraph ( t, tuple3(a, b, c) ) {
    tVar ( int, x );
    tVar ( int, y );
    tVar ( int, z );

    tGetId ( loop[0] ); // label
    tFor ( x, 0, MATRIXSIZE - 1 ) {
      tGetId ( loop[1] ); // label
      tFor ( z, 0, MATRIXSIZE - 1 ) {
        tGetId ( loop[2] ); // label
        tFor ( y, 0, MATRIXSIZE - 1 ) {
          c[x][y] += a[x][z] * b[z][y];
        }}}}}

main () {
  int bestTime; int bestSize = 0;
  for (int tsz = 4; tsz <= MATRIXSIZE; ++tsz) {
    int trip3 = { tsz, tsz, tsz };
    TaskLoopIdentifier loop[3];
    mm_TaskGraph MM;

    taskMatrixMult(loop, MM);
    interchangeLoops(loop[1], loop[2]);
    tileLoop(3, &loop[0], trip3);

    MM.compile(TaskGraph::ICC);

    tt3 = time_function();
    MM.execute(A, B, C);
    time = time_function()-tt3;

    if (time < bestTime || bestSize == 0) {
      bestTime = time; bestSize = tsz;
    }}}
```

--\-- MM.cc (C++)--L24--All--------------------

```
extern void taskGraph_1(void **params)
{
  float (*a)[512];
  float (*b)[512];
  float (*c)[512];
  int i;
  int j;
  int k;
  int j_tile;
  int k_tile;

  a = *params;
  b = params[1];
  c = params[2];
  for (i = 0; i <= 511; i++)
    for (j_tile = 0; j_tile <= 511; j_tile += 64)
      for (k_tile = 0; k_tile <= 511; k_tile += 64)
        for (j = j_tile;
                j <= min(511, 63 + j_tile); j++)
          for (k = max(0, k_tile);
                  k <= min(511, 63 + k_tile); k++)
            c[i][k] = c[i][k] + a[i][j] * b[j][k];
}
```

- Generated code

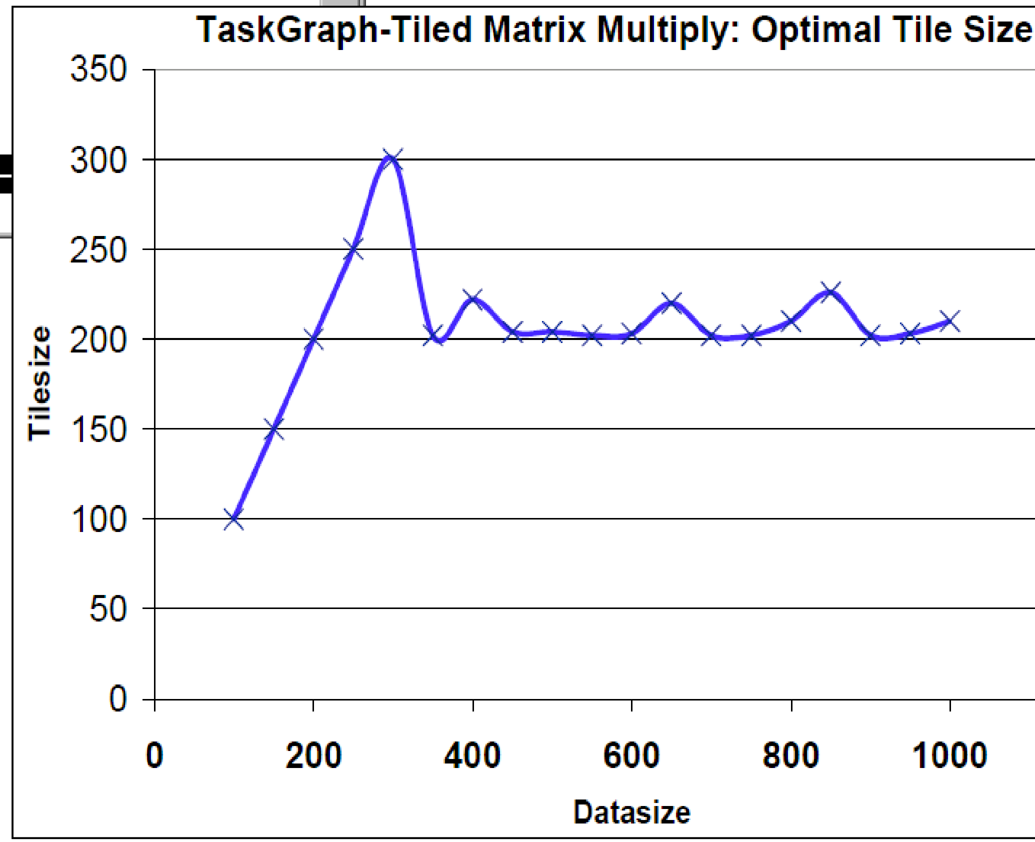
  (Slightly tidied)

Adapting to platform/resources

```
int bestTime;
int bestSize = 0;
for (int tsz = 4; tsz <= MATRIXSIZE; ++tsz) {
  int trip3 = { tsz, tsz, tsz };
  TaskLoopIdentifier loop[3];
  TaskGraph MM;
  taskMatrixMult(loop, MM);
  interchangeLoops(loop[1], loop[2]);
  tileLoop(3, &loop[0], trip3);
  MM.compile(TaskGraph::ICC, false);
  tt3 = time_function();
  MM.execute("A",A, "B",B, "C",C, NULL);
  time = time_function()-tt3;
  if (time < bestTime || bestSize == 0) {
    bestTime = time; bestSize = tsz;
  }
}
```

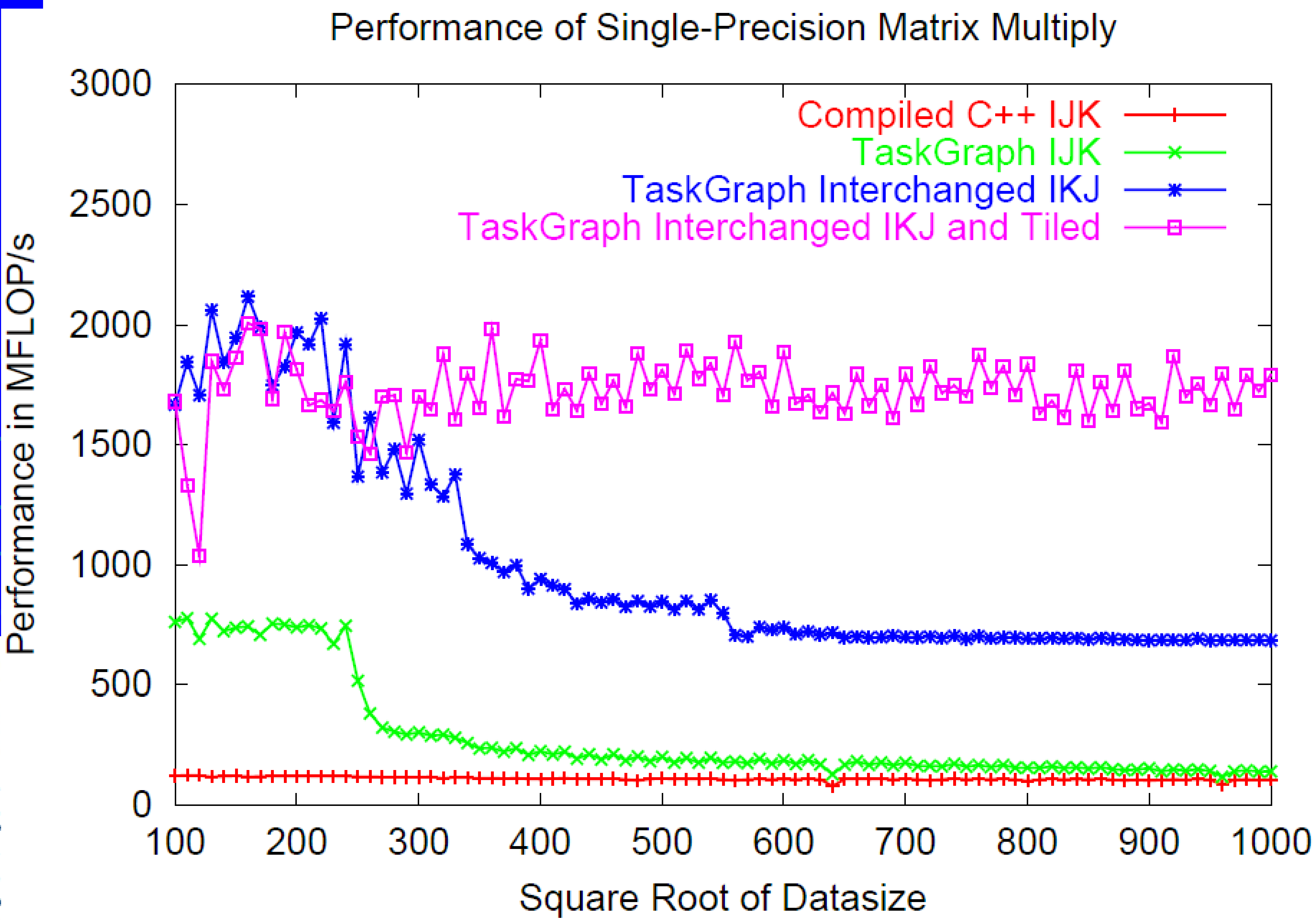
--\-- IterativeMM.cc   (C++)--L2-- 2%---

- We can program a search for the best implementation for our particular problem size, on our particular hardware

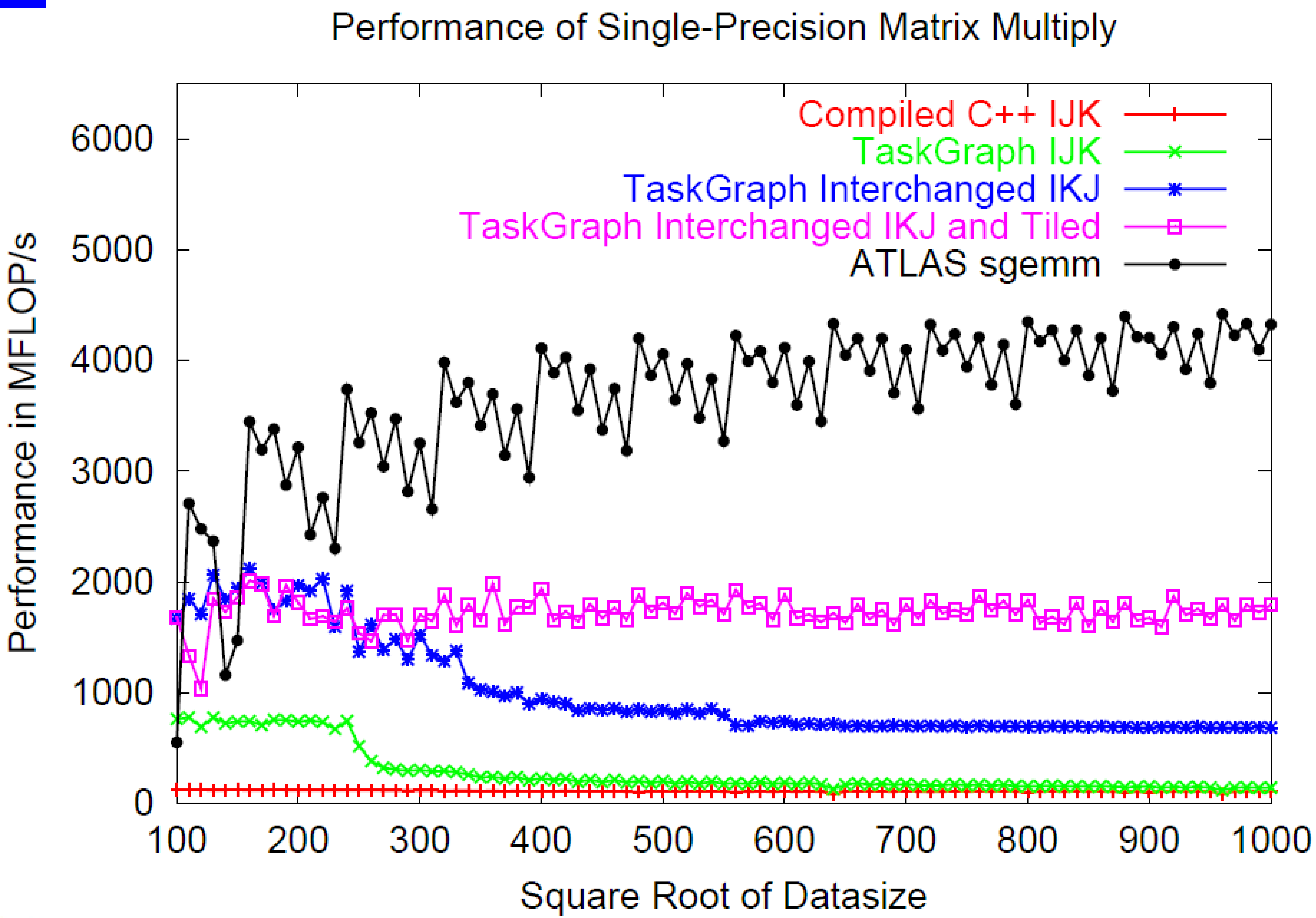


On Pentium 4-M, 1.8 GHz, 512KB L2 cache, 256 MB, running Linux 2.4 and icc 7.1.

Adapting to platform/resources



Adapting to platform/resources



Potential for user-directed restructuring

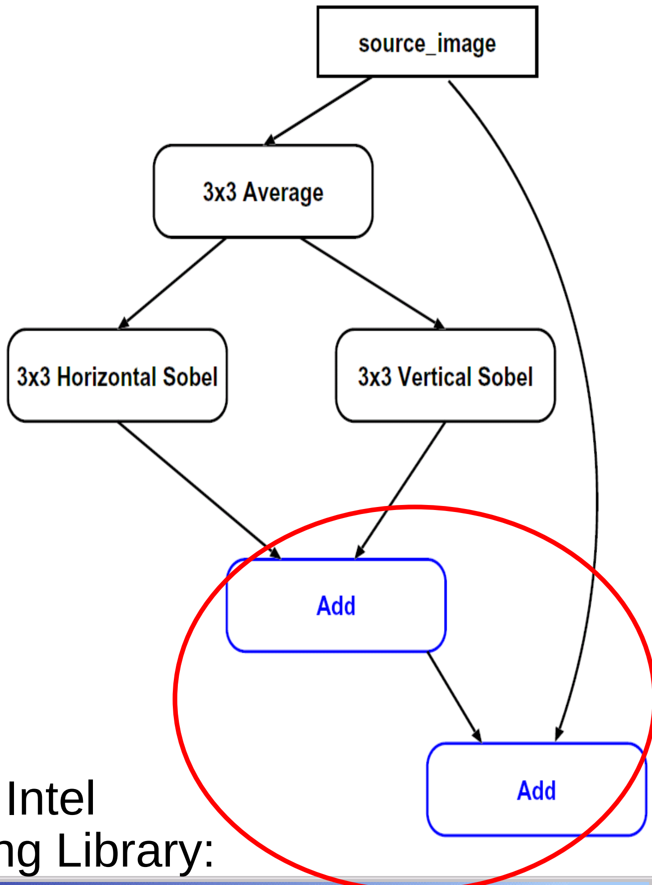
- Programmer controls application of sophisticated transformations
- Performance benefits can be large – in this example >8x
- Different target architectures and problem sizes need different combinations of optimisations
  - ijk or ikj?
  - Hierarchical tiling
  - 2d or 3d?
  - Copy reused submatrix into contiguous memory?
- Matrix multiply is a *simple* example

Cross-component loop fusion

emacs@SECONDSELF

Buffers Files Tools Edit Search Mule C++ Help

```
  TaskGraph T;
  taskgraph( T ) {
    unsigned int ds[] = fsz, szg;
    tParameter(tArrayFromList(float, dstimg, 2, ds));
    tParameter(tArrayFromList(float, srcimg, 2, ds));
    tArrayFromList( float, blur , 2, ds );
    // ...
    instantiateBlur (blur , srcimg, i , j , sz, sz , 3);
    instantiateSobelHoriz(horiz , blur , i , j , sz, sz);
    instantiateSobelVert (vert , blur , i , j , sz, sz);
    instantiateAdd(both, vert , horiz , i , j , sz, sz);
    instantiateAdd(dstimg, blur , both, i , j , sz, sz);
  }
  T.applyOptimisation ("fusion");
  T.compile(TaskGraph::ICC, true);
  T.execute("dstimg", result , "srcimg" , image, NULL);
}
--\--  Filter.cc          (C++)--L10--Bot-----------------
```



- Image processing example
- Blur, edge-detection filters then sum with original image
- Final two additions using Intel Performance Programming Library:

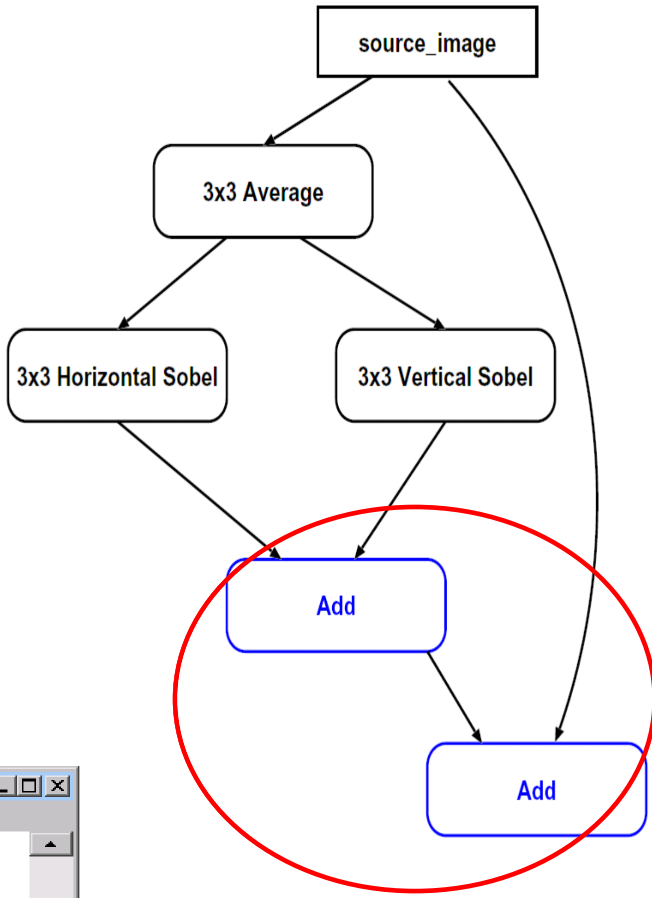
emacs@SECONDSELF

Buffers Files Tools Edit Search Mule C++ Help

```
  // Ipp Domain Specific Library
  ippiAdd_32f_C1R( horiz, length , vert , length,
                   both, length , whole );
  ippiAdd_32f_C1R( image, length, both, length,
                   result , length , whole );

--\--  FilterIPP.cc       (C++)--L5--All----------
```

Cross-component loop fusion

```
emacs@SECONDSELF
Buffers Files Tools Edit Search Mule C++ Help
// TaskGraph Generated Code
  for ( i = 0; i <= 1199; i++) {
    for ( j = 0; j <= 1599; j++) {
      both[ i ][ j ] = vert [ i ][ j ] + horiz [ i ][ j ];
    }
  }
  for ( i = 0; i <= 1199; i++) {
    for ( j = 0; j <= 1599; j++) {
      tgimage[i ][ j ] = blur [ i ][ j ] + both[ i ][ j ];
    }
  }
}
--\--  FilterGenerated.cc      (C++)--L2--Bot-------------
```

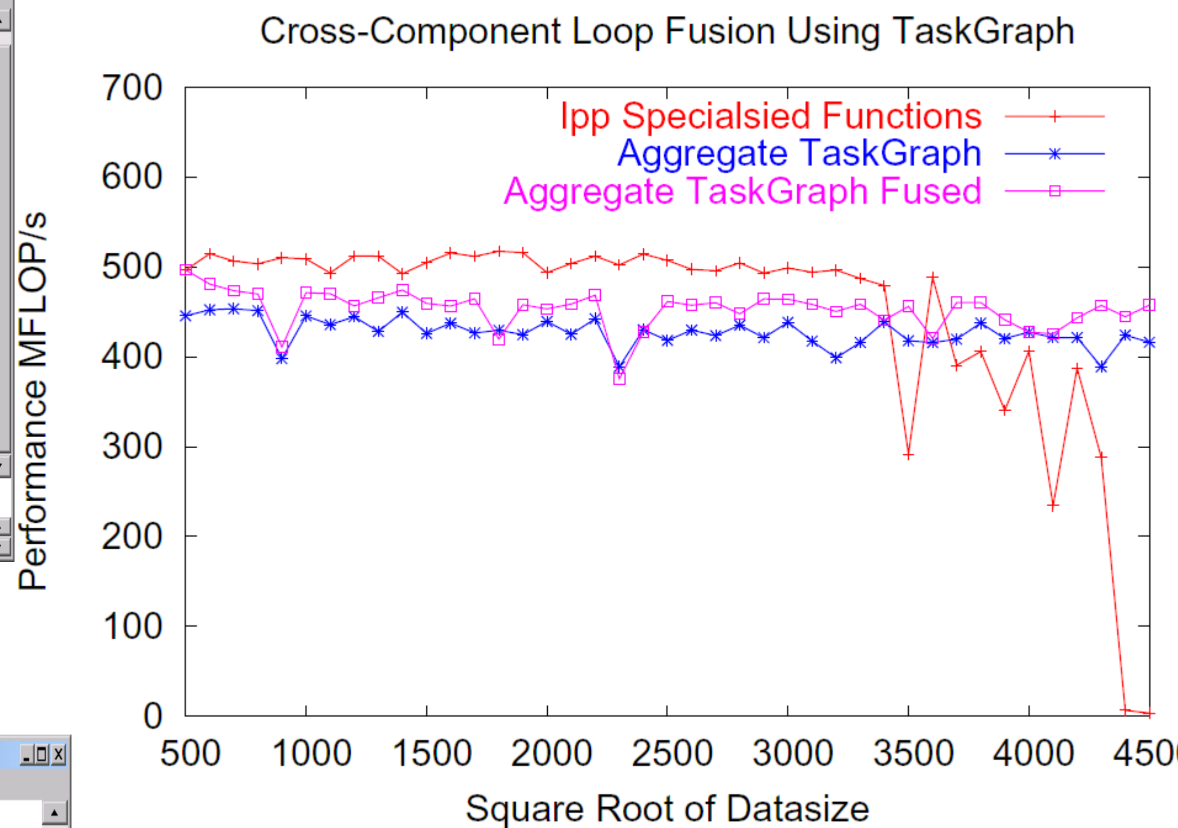


- After loop fusion:

```
emacs@SECONDSELF
Buffers Files Tools Edit Search Mule C++ Help
// TaskGraph Optimised Generated Code
  for ( i = 0; i <= 1199; i++) {
    for ( j = 0; j <= 1599; j++) {
      both[ i ][ j ] = vert [ i ][ j ] + horiz [ i ][ j ];
      tgimage[i ][ j ] = blur [ i ][ j ] + both[ i ][ j ];
    }
  }
}
--\--  FilterGenerated.cc      (C++)--L16--Bot------------
```

Cross-component loop fusion

```
// TaskGraph Generated Code
for ( i = 0; i <= 1199; i++) {
  for ( j = 0; j <= 1599; j++) {
    both[ i ][ j ] = vert [ i ][ j ] + horiz [ i ][ j ];
  }
}
for ( i = 0; i <= 1199; i++) {
  for ( j = 0; j <= 1599; j++) {
    tgimage[i ][ j ] = blur [ i ][ j ] + both[ i ][ j ];
  }
}
}
--\--  FilterGenerated.cc      (C++)--L2--Bot------------
```



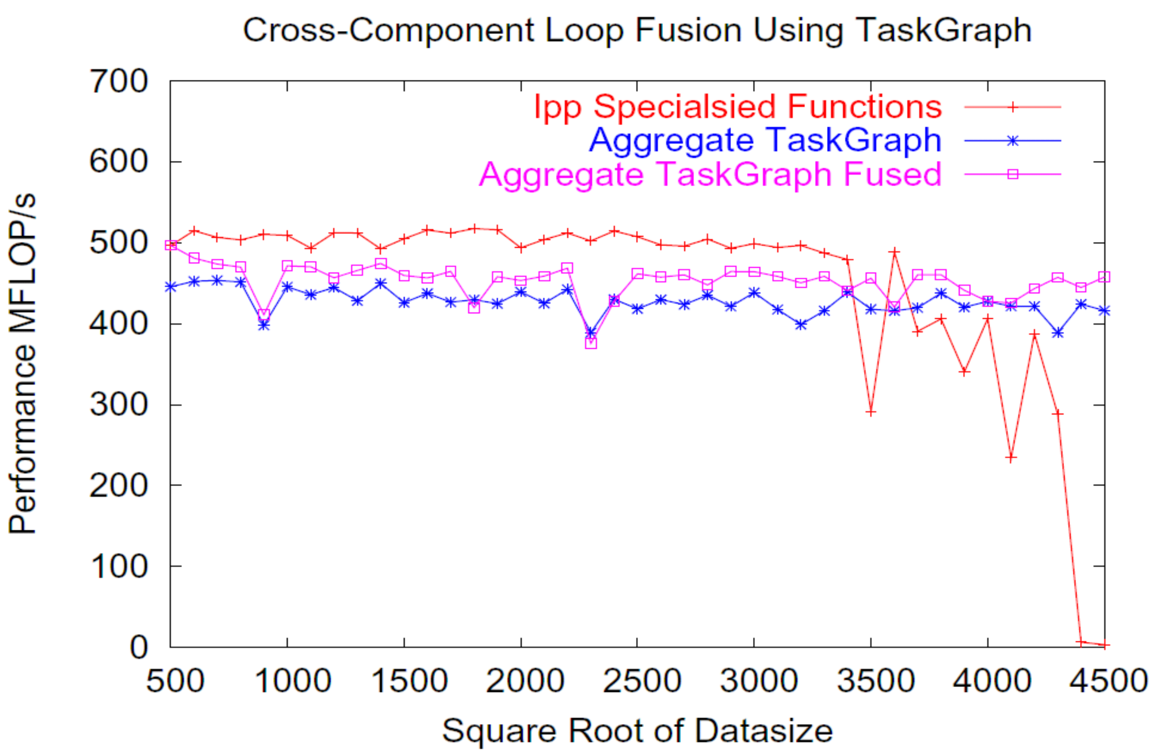
After loop fusion:

```
// TaskGraph Optimised Generated Code
for ( i = 0; i <= 1199; i++) {
  for ( j = 0; j <= 1599; j++) {
    both[ i ][ j ] = vert [ i ][ j ] + horiz [ i ][ j ];
    tgimage[i ][ j ] = blur [ i ][ j ] + both[ i ][ j ];
  }
}
}
--\--  FilterGenerated.cc      (C++)--L16--Bot------------
```

- Simple fusion leads to small improvement
- Beats Intel library only on large images
- Further fusion opportunities require skewing/retiming

Performance metadata informs cross-component optimisation

- We know we *can* fuse the two image addition loops
- However our performance results show this is only sometimes faster
  - For small images it's faster to call the Intel Performance library functions one-at-a-time
  - On this machine, fusion is a huge benefit – but only for images > 4000x4000



- How can we tell what to do?
  - Could use static rule "on a Pentium4 fuse if size >4000"
  - Could experiment at runtime, measure whether fusion is faster, roll-back if not
  - Could use hardware performance counters – if TLB and L2 cache miss rate are low, fusion unlikely to win

Conclusions

- TaskGraph library delivers run-time code generation (as found in `C, Jak, MetaOCaml etc) as a library, rather than a language extension
- SUIF offers the metaprogrammer full power of a restructuring compiler
- Aggressive compiler techniques can be especially effective:
  - The TaskGraph language is simple and clean
  - TaskGraphs are usually small
  - Compilation effort can be directed by the programmer
  - Domain knowledge can direct the focus and selection of optimisations
  - Programmers can build and share domain-specific optimisation components
- Domain-specific optimisation components have lots of potential

Restructuring loops by metaprogramming

- The taskgraph library is still at the prototype stage
- We have ambitious plans for this work:
  - Combining specialisation with dependence analysis and restructuring
    - cf inspector-executor
  - Domain-specific optimisation components
    - Build collection of optimisation components specialised to computational kernels of particular kinds
    - Eg stencil loops (Jacobi, red-black, Gauss-Seidel etc)
  - Combine
    - domain-specific information (eg algebraic properties of tensor operators)
    - Problem-specific information (eg sizes and shapes of data)
    - Context-specific information (the application's control and data dependence structure)

TaskGraph – open issues...

- Types
  - TaskGraph library currently limited to scalars+arrays. How can we use calling program's data types, in an efficient and type safe way?
  - How can we check that the generated code is being used in a safe way?
- Compilation overhead
  - Building and compiling small code fragments takes ~100ms. Mostly in C compiler (not TGL or SUIF). This is a major problem in some applications, eg JIT
- Metaprogramming API
  - Much more work is needed on designing a flexible representation of the dependence information we have (or need) about a TaskGraph (eg Dan Quinlan's ROSE)
  - Fundamental issue is to make metadata smaller than the data
- Introspection and naming
  - Need to think more about how a metaprogrammer refers to the internal structures of the subject code – "which loop did I mean?"

Domain-specific optimisation – open issues

- Domain-specific *optimisation* is surprisingly hard to find
- Domain-specific information is hard to use
  - How to capture a software component's characteristics, so that the component can be optimised to its context (or mode) of use.
  - How to represent the space of possible optimisation alternatives for a component, so that the best combination of optimisations can be chosen when the component is used.
  - How to represent the relevant internal structure of a component so that domain-specific optimisations can be implemented at a sufficiently abstract level to be re-usable and easy to construct.

Components for performance programming

- Component's functional interface
- Component's adaptation interface
- Component metadata
  - Characterizes how the component can adapt
  - Provides performance model
  - Provides elements from which composite optimisation formulation can be assembled
- Composition metaprogramming
  - Uses components' metadata to find optimal composite configuration
  - Uses component adaptation interfaces to implement it
  - May also deploy and use instrumentation to refine its decision

Imperial College London