How to handle code generation on modern processors architectures?

Or another way to say “Yes we can”

Henri-Pierre Charles

Université de Versailles Saint-Quentin en Yvelines

6 mars 2009
Initial problem: A car has fell into the harbour
Solution: a big truck can remove it
Hoops: Second problem
A metaphor
The compilation chain
HPBCG: High Performance Binary Code Generator
What reasons I have to use this damn thing?

A problem to solve (new or old?)
Old solution: a bigger truck
A metaphor

The compilation chain

HPBCG : High Performance Binary Code Generator

What reasons I have to use this damn thing?

Remove the car

Henri-Pierre Charles

How to handle code generation on modern processors architecture
Then remove the truck
A metaphor
The compilation chain
HPBCG : High Performance Binary Code Generator
What reasons I have to use this damn thing?

Hoops : Third problem
Can we always solve new problem with old solutions?
A metaphor

The compilation chain

HPBCG: High Performance Binary Code Generator

What reasons I have to use this damn thing?

The normal way

Blue gene IBM xlc compiler documentation

Source code is not high level

The “Titanic” effect 1/2

Compiler dilemma

Source code is supposed to be “high level”

Runnable code is supposed to be “low level”

Optimization is supposed to be “data independant”

The target machine is supposed to “be known”
## “Low level instruction” (Itanium ISA)

### psad — Parallel Sum of Absolute Difference

<table>
<thead>
<tr>
<th>Format:</th>
<th>( qr ) <code>psad</code>  <code>r1</code> <code>r2</code> <code>r3</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The unsigned 8-bit elements of GR <code>r3</code> are subtracted from the unsigned 8-bit elements of GR <code>r2</code>. The absolute value of each difference is accumulated across the elements and placed in GR <code>r1</code>.</td>
</tr>
</tbody>
</table>

**Figure 2-38. Parallel Sum of Absolute Difference Example**

Compilers designer are **always in late** in race with processors architect.
“Low level instruction” (Power ISA)

- PPC440 FP2 Architecture
- Blue Gene L architecture
  - 32 Pair of FP registers (Seen as Complex or vector)
  - Special instructions `fxcxnpma t, a, b, c`
    \[ t_p = (A_s \times C_s - B_P),\ t_s = A_s \times C_P + B_S \]
- **BUT**: C has no standard support for complex number (intrinsic or libraries are not portable)
“Exploiting the Dual Floating Point Units in Blue Gene/L”

**Future directions**

IBM plans to continue to address performance of dual FPU code in future updates and releases. Improvements in the SIMD framework will also benefit BlueGene. We expect that this will lead to better exploitation of the dual FPU.

**Summary**

The presence of a second FPU on the BG/L processors potentially allows double the performance on floating point algorithms over just using a single FPU. The ability of the IBM XL compilers to automatically use the dual FPU unit depends strongly on the properties of the source code. The more regular the accesses to floating point data, the more the compiler is able to exploit the dual FPU. Examples of regular access include matrix multiplication and vector processing. This paper has described the implementation of the dual FPU in BlueGene/L and some limitations of automatic compiler exploitation of this dual FPU. Our long term goal is to ensure that using the dual FPU will be no slower than single FPU code. This may not be achievable, due to the extra versioning necessary for alignment or aliasing checks, but the overhead should be minimized.
Source code is not high level

ARM teach how to write code with architecture in mind

How does the programmer’s model affect C?

- Choose the right algorithm
- Make best use of the instruction set
- Make good use of available memory types
- Lay out data structures carefully
- Write C with the underlying architecture in mind
The "Titanic" effect 1/2

Data set modify performance application during run-time

Using a FreeBOX (ADSL Modem and TV/IP broadcasting), my home computer (Intel Celeron 1.70GHz) play correctly France3 channel (which use MPEG TS video format) but cannot play RTL9 (X264 video format)

Optimization is not data independent.
Data set modify performance application during run-time

Extract from MPEG group benchmarks

Optimization is not data independant
Ask for program!

What are speed variation for this program:

```c
int i;
for (i = 0; i < N; ++i)
{
    int j;
    dest[i] = 0;
    for (j = 0; j < N; ++j)
        dest[i] += src[j] * m[i][j];
}
```

Compiler, data size, target processor, available parallelism, data type, memory location, operating system, ...
Size Matter (… of Data size, what do you think?)

Loop size (value of $N$)

$10^1$ Multimedia kernel: Full loop unroll, instruction scheduling, memory caches access, ...

$10^2/10^3$/ Scientific code: loop unroll, loop conversion, data prefetching

$10^6$ Multimedia flux: multithreading

$10^{10}$ and more High level parallelism: MPI / Grid / Cloud, ...

$N$ is generally a parameter only known at run-time. Profiling and iterative compilation does not help.

Compilation strategies are complex and are application domain specific

"Titanic" effet
Let’s start from the low level

**BECAUSE**

- this is the main constraint (which come from industry)
- they are many optimization opportunities
- it give the programming model (which is the main constraint for compiler)
- “The advent of just-in-time compilation for languages (such as Java) blurs the distinction between compile time and runtime, opening up new opportunities for program optimization based on dynamically computed program values. As parallel client-side applications emerge, runtime dependence checking and optimization are likely to be essential for optimizing programs that manipulate dynamic data structures” *(Compiler Research: the next 50 Years; Février 2009; Mary Hall, David Padua and Keshav Pingali; Communication of the ACM)*
What do I need

Objective

- Mix “constant” data & binary program
- Full ISA description (multimedia instructions, baroque arithmetics, vector instructions)
- Portable code generation (mix multiple ISA)

Tool Needed

- FAST architecture description
- FAST code generation
Cell description

Henri-Pierre Charles

How to handle code generation on modern processors architecture
A metaphor

The compilation chain

HPBCG : High Performance Binary Code Generator

What reasons I have to use this damn thing ?

Let's start from the low level

What do I need

Cell description

Cell PPU

Synergistic processor element

- Vectorial instruction set
- Small memory
- spu-gcc
- No direct memory access
- No system

Cell PPU

Power4 core

- Power4 instruction set + ALTIVEC
- gcc
- Classical processor

“Data choreography” & “Code placement”
unsigned char code[] =
{
    0x78, 0x81, 0x01, 0x83,   /* mpy $3,$3,$4 */
    0x35, 0x00, 0x00, 0x00   /* bi $0 */
};
..
typedef int (*pfiii)(int, int);
pfiii Sad = (pfiii) code;
int result = Sad(2, 21);
What time is it?

**Thinking time**  Write the code generator generator, write a “compilette”.

**Install time**  Extract the code generator from a basic description to a mid level description

**Static compile time**  Translate compilette to mid level

**Start time**  Parameter analysis, “pre-run compile time”

**Run time**  Generate the code, then run it, “on the edge compile-time”
A metaphor
The compilation chain
HPBCG: High Performance Binary Code Generator
What reasons I have to use this damn thing?

Let’s start from the low level
What do I need
Cell description
Compil chain
Ask for the program

Documentation example

---

IBM

Instruction Set Architecture
Synergistic Processor Unit

Multiply Immediate

\[ \text{mpyi} \quad \text{rt,ra,value} \]

For each of four word slots:
- The signed value in the I10 field is multiplied by the value in the rightmost 16 bits of register RA.
- The resulting product is placed in register RT.
Describe the architecture. Example cell-spu:

cell 32

..../..

# Integer and Logical Instructions

01111000100  r3_7  r2_7  r1_7  | mpy    r1, r2, r3
01111001100  r3_7  r2_7  r1_7  | mpyu   r1, r2, r3
01110100     i1_9-0  r2_7  r1_7  | mpyi    r1, r2, i1
01110101     i1_9-0  r2_7  r1_7  | mpyui   r1, r2, i1
1100         r1_7  r3_7  r2_7  r4_7  | mpya    r1, r2, r3, r4

../../

Easy to do from the processor documentation
Parse the ISA description and generate

- the macro instruction generator:

```c
#define mpyi_iRRI(r1,r2,i1) 
   ADDINSN(((((( LENOK(116, 8) )<< 10 
         | LENOK((i1 & 0x3ff), 10) )<< 7 
         | LENOK(r2, 7) )<< 7 
         | LENOK(r1, 7) ))
```

- the function instruction generator:

```c
void mpyi_iRRI (int r1,int r2,int i1){
   ADDINSN(((((( LENOK(116, 8) )<< 10 
         | LENOK((i1 & 0x3ff), 10) )<< 7 
         | LENOK(r2, 7) )<< 7 
         | LENOK(r1, 7) )));

#ifdef ASM_DEBUG
printf("%p : %s%s 0x%X\n", asm_pc, "mpyi", "_iRRI", *(asm_pc-1));
#endif /* ASM_DEBUG */
}
```
Write a “compilette” (thinking time)

```c
pifi multiplyCompile(int multiplyValue)
{
    insn *code = (insn *)_malloc_align(1024, 7);
    printf("Code generation for multiply value %d\n", multiplyValue);
    
    .org code
    mpyi $3, $3, (multiplyValue)
    bi $lr
    
    printf("Code generated\n");
    return (pifi)code;
}
```
Use the "compilette"

/* Generate binary code */
multiplyFunc = multiplyCompile(atoi(argv[1]));
for (i = 1; i < 11; ++i)
    printf("%3d ", i);
printf("\n");
for (i = 1; i < 11; ++i)
    printf("%3d ", multiplyFunc(i));
printf("\n");
Generate the code generator (Static compile time)

```
hpbcg simple-multiply-cell.hg > simple-multiply-cell.c
cc -Wall ../.. simple-multiply-cell.c -o simple-multiply-cell
spu-gcc ../.. -o simple-worker-cell simple-worker-cell.c
```
Run the code generator (Run time)

```
turner:simple-multiply/>./simple-multiply-cell 6
Code generation for multiply value 6
Code generated
  1  2  3  4  5  6  7  8  9  10
  6 12 18 24 30 36 42 48 54 60

turner:simple-multiply/>./simple-multiply-cell 7
Code generation for multiply value 7
Code generated
  1  2  3  4  5  6  7  8  9  10
  7 14 21 28 35 42 49 56 63 70

turner:simple-multiply/>./simple-multiply-cell 14
Code generation for multiply value 14
```

Henri-Pierre Charles

How to handle code generation on modern processors architectures?
Ask for the program

- [ ] "http://hpcbg.org/
- Support for:
  - **power** power4, altivec, cell, FP2
  - **itanium** full isa but not scheduling
  - **arm** preliminary
- Samples codes
- BSD like licence
Code specialization

Let’s remove one function parameter:
void Function(float *A, float *B, int size /*, int stride*/) {
    int i;
    for (i = 0; i < size; i++)
}
Then compile with:

```shell
../../
icc -Dstride=5 -S -O Example.c -o Example.s.5
icc -Dstride=6 -S -O Example.c -o Example.s.6
icc -Dstride=7 -S -O Example.c -o Example.s.7
icc -Dstride=8 -S -O Example.c -o Example.s.8
../../
diff Example.s.[57]
```
A metaphor
The compilation chain
HPBCG : High Performance Binary Code Generator
What reasons I have to use this damn thing?

Using specialization
Kernel optimization
General goal :
TODO list

Specialization scheme : compile time

C Code
cc −DVariable=Value

Few templates
C Code
cc −DVariable=Value

Runtime
Specializer
+

Henri-Pierre Charles
How to handle code generation on modern processors architectures
Specialization scheme: run-time

Data width

- Code
- Library
- Data

DynamicSp

Henri-Pierre Charles

How to handle code generation on modern processors architectures?
Specialization scheme: compile-time
Using specialization

**Static**  Generate all different versions + a planner (The most often used version and the fallback version)

**Dynamic**  Generate 1 template + a dynamic specializer

**Hybrid**  Generate multiple templates + a cache + a specializer
Multimedia Compilettes

Example: 20 loads, 4 stores, 16 mult., 12 add.
Integer or FP computation

#define T 4

void dotProduct(Matrix m, Vector src, Vector dest)
{
    int i;
    for (i = 0; i < T; ++i)
    {
        int j;
        dest[i] = 0;
        for (j = 0; j < T; ++j)
        {
            dest[i] += src[j] * m[i][j];
        }
    }
}
Multimedia Compilettes

4 loads, 4 stores, 3 mult., 3 add.

/* Translation matrix : 
 * 1 0 0 0
 * 0 1 0 0
 * 0 0 1 0
 * Tx Ty Tz 1 */

void dotProduct(Vector src, Vector dest)
{
    dest[0]= src[0];
    dest[1]= src[1];
    dest[2]= src[2];
}
“Compilettes”

A metaphor
The compilation chain
HPBCG : High Performance Binary Code Generator
What reasons I have to use this damn thing?

Using specialization
Kernel optimization
General goal :
TODO list

"Compilettes"

Source code → Intermediate code → Assembly code → Binary code → Runnable code

Compiler → Assembler → Loader → System

Algorithmic optimizer

Code generation

User → Data → Parameter

Assembler → Loader

Henri-Pierre Charles

How to handle code generation on modern processors architectures?
How does it works?

Data width

Code

Library

Data

Compilette

Henri-Pierre Charles

How to handle code generation on modern processors architectures
Compilettes / Results

Obtained goals:

1. Matrix vector vectorisation
2. Matrix vector optimization
3. Image filtering optimisation (using multimedia instructions)
4. X264 video encoder optimization (data alignment, multimedia instructions)
5. OpenGL off screen optimization (vectorization, multimedia instructions, optimization)

Multi target: Sparc Itanium, Power (Power & Altivec), Cell (SPU & PPU), CCG / HPBCG
General goal: experiment new compilation process

A metaphor
The compilation chain
HPBCG: High Performance Binary Code Generator
What reasons I have to use this damn thing?

Using specialization
Kernel optimization
General goal: TODO list

Henri-Pierre Charles
How to handle code generation on modern processors architectures?
TODO list

- Go high level
  - Find a “compilette” generator (gcc-LTO project, C++, ..)
  - Use it in the large (STAPL)
- Find a new position!