An approach to Performance and Bottleneck Analysis

Sverre Jarp
CERN openlab

Summer student lecture 2006
1 August 2006
AGENDA

- Introduction
- Path to Optimization
- Basics
- Hardware Review
- More on compilers
- Conclusion
INTRODUCTION
Initial Question

• How come we (too) often end up in the following situation?

Why the heck doesn’t it perform as it should!!!
Before we start

• This is an effort to pass in review a somewhat “systematic approach” to tuning and bottleneck analysis
  – Main focus is on understanding the “spiral to success”

• The introduction of the elements is done “top-down”

• But, it is important to understand that in real-life, this is rarely the case
PATH TO OPTIMIZATION
Path to optimization

Source

Execution Results

Compiler

Software Design

Platform Run
Step 0: Correctness

• **Before undertaking any tuning effort**
  – Excellent regression/correctness tests
    • For all critical algorithms, all important use cases
  – Otherwise,
    • Too many tuning efforts get left by the wayside

• Which options will work?
  – “IPF_fp_relax”
  – “ansi_alias”
  – “ffast-math”

In an case, needed for the basic development/maintenance effort

No point in speeding up an incorrect program!
Step 1: Application design

- **Regular reviews of the design (globally or partially)**
  - Data structures
    - Arrays; structs; data members
  - Choice of algorithms
    - Accuracy, robustness, rapidity
  - Design of classes
    - Domain decomposition
    - Hierarchy
    - Interrelationship
- **Is there a time gap?**
  - Design ↔ Today’s microprocessor (tomorrow’s ?)
    - Did we design for low ILP, small caches, single core,…. ?

The arrival of multi-/many-core may force partial redesign of many applications
Step 2: Implementation aspects

- Review all aspects of implementation
  - Choice of language (Fortran, C, C++, Java, …)
  - Use of language features
    - Templates (STL with maps, lists, etc.)
  - Precision of data (FLP)
    - Single, double, double extended
      - Intermediate calculations
      - Stored results
  - Code split between .cpp and .h files
  - Aggregation or decomposition?
  - Reliance on preprocessor
  - Platform dependencies
    - Such as endianness
  - Reliance on external libraries
    - Smartheap, Math kernel/vector libraries, etc.

Correct organization of source can greatly impact the application’s efficiency
Step 3: Compiler/compilation

- **Access to the best compiler**
  - On many platforms we have a limited choice
    - IA-64/Linus or x86/MacOS: Intel or GNU (others coming?)
  - But, it is worth trying both (or all):
    - Mix and match (thanks to common ABI)?
    - Inform the other camp when they are behind
  - Upgrade to latest versions regularly
  - Choose from hundreds of flags

- **Build procedure**
  - One class at a time?
  - Archive/shared libraries?
  - Monolithic executable or dynamic loading?

- **And (to a large extent)**
  - Machine code is chosen for you

Note that x86-64 is a very healthy clean-up of the too-often extended x86 architecture
Step 4: Platform

- **The best hardware for the job**
  - Manufacturer
  - Server type
    - Entry, mid-range, large SMP, NUMA, etc.
  - Processor characteristics
    - Single core, Dual core, Quad core (coming)
    - Frequency, cache sizes and levels
  - Further (important) factors
    - Bus speed
    - Memory speed
  - Price/performance ratio

Richest choice is found inside the x86 eco-system.
In the end: Execution Results

Source Code

Compiler
Machine Code

Execution Results

Platform

Design of Data Structures and Algorithms
Back to our cartoon

• As already said, first of all, we must guarantee correctness

• If we are unhappy with the performance
  – … and by the way, how do we know when to be happy?

• We need to look around
  – Since the culprit can be anywhere
Where to look?

Design of Data Structures and Algorithms

Source Code

Execution Results

Compiler Machine Code

Platform
THE BASICS
Need a good tool set

- **My recommendation**
  - Integrated Development Environment (IDE) w/ integrated Performance Analyzer
    - Visual Studio + VTUNE (Windows)
    - Eclipse + VTUNE (Linux)
    - XCODE + Shark (MacOS)
    - ...

- **Also, other packages**
  - Valgrind (Linux x86, x86-64)
  - Qttools (IPF)
  - Pfmon, perfsuite, caliper, oprofile, TAU

Too many different tools may be counterproductive!
### Price_out_impl (mcf)

#### VTUNE screenshot:

<table>
<thead>
<tr>
<th>Address</th>
<th>Line</th>
<th>Clockticks</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:03CA</td>
<td>246</td>
<td>1426</td>
<td>while( arcin )</td>
</tr>
<tr>
<td></td>
<td>247</td>
<td></td>
<td>{</td>
</tr>
<tr>
<td>01:03F0</td>
<td>248</td>
<td>2329</td>
<td>tail = arcin-&gt;tail;</td>
</tr>
<tr>
<td>01:03F2</td>
<td>250</td>
<td>323207</td>
<td>if( tail-&gt;time + arcin-&gt;org_cost &gt; latest )</td>
</tr>
<tr>
<td></td>
<td>251</td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>252</td>
<td>1485</td>
<td>arcin = (arc_t *)tail-&gt;mark;</td>
</tr>
<tr>
<td></td>
<td>253</td>
<td></td>
<td>continue;</td>
</tr>
<tr>
<td></td>
<td>254</td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>01:0410</td>
<td>256</td>
<td>657</td>
<td>red_cost = compute_red_cost( arc_cost, tail, head_potential );</td>
</tr>
<tr>
<td>01:0424</td>
<td>258</td>
<td>2233</td>
<td>if( red_cost &lt; 0 )</td>
</tr>
<tr>
<td></td>
<td>259</td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>260</td>
<td>296</td>
<td>if( new_arcs &lt; MAX_NEW_ARCS )</td>
</tr>
<tr>
<td></td>
<td>261</td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>262</td>
<td></td>
<td>insert_new_arc( arcnew, new_arcs, tail, head,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>arc_cost, red_cost );&quot;</td>
</tr>
<tr>
<td></td>
<td>263</td>
<td>39</td>
<td>new_arcs++;</td>
</tr>
<tr>
<td></td>
<td>264</td>
<td>16</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>265</td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>01:04FE</td>
<td>266</td>
<td>301</td>
<td>else if( (cost_t)arcnew[0].flow &gt; red_cost )</td>
</tr>
<tr>
<td></td>
<td>267</td>
<td></td>
<td>replace_weaker_arc( arcnew, tail, head,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>arc_cost, red_cost );</td>
</tr>
<tr>
<td>01:050B</td>
<td>268</td>
<td>65</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>269</td>
<td></td>
<td>arcin = (arc_t *)tail-&gt;mark;</td>
</tr>
<tr>
<td>01:042C</td>
<td>271</td>
<td>1664</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>272</td>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
Assembly language literacy

• The language spoken by the processor is
  – MACHINE CODE !!

• To understand it, we need what I call “Assembler awareness”:
  – Looking into compiler-generated code, there may be a need to:
    • Modify (repeatedly) the HLL code (or compiler options) and inspect
      the result
    • When available, add inline assembly or intrinsics for localized
      impact
  – Today, we are not dealing with the case of writing Assembly
    code
    • But the issues are the same
• It may be necessary to read the machine code directly

```c
Bool_t TGeoCone::Contains(Double_t *point) const 
{
   // test if point is inside this cone
   if (TMath::Abs(point[2]) > fDz) return kFALSE;

   Double_t r2 = point[0]*point[0] + point[1]*point[1];
   Double_t rl = 0.5*(fRmin2*(point[2] + fDz) + fRmin1*(fDz-point[2]))/fDz;
   Double_t rh = 0.5*(fRmax2*(point[2] + fDz) + fRmax1*(fDz-point[2]))/fDz;
   if ((r2<rl*rl) || (r2>rh*rh)) return kFALSE;
   return kTRUE;
}
```

```assembly
_ZNK8TGeoCone8ContainsEPd:
[.LFB1785:]
   .prologue
   .body
   .mmi
   adds r14 = 16, r33
   adds r15 = 16, r32
   adds r16 = 32, r32
   .mmi
   adds r17 = 24, r32
   adds r18 = 40, r32
   adds r32 = 8, r32  ;;
   .mmi
   ldfd f11 = [r14]
   ldfd f15 = [r32]
   mov r8 = r0  ;;
   .mfb
   fcmp.ge p6, p7 = f11, f0
   .mfi
   mov f6 = f11  ;;
   .mmf
   (p7) fneg f6 = f11  ;;
   .mmf
   fcmp.gt p6, p7 = f6, f15; ;
   .bbb
   (p6) br.ret.dptk.many rp
(snip)
```
Amdahl’s Law

- The incompressible part ends up dominating:

  100%

  10%  20%  20%  20%  30%

  Total speedup is “only”:  \(\frac{100}{80} = 1.25\)

  Great job, Sverre: 3x!
### Test40: Physics simulation job

#### Command:
```
./test40icc80O2fz
```

#### Flat profile of CPU_CYCLES in test40icc80O2fz-pid24686-cpu0.hist#0: Each histogram sample counts as 1.00034m seconds

<table>
<thead>
<tr>
<th>% time</th>
<th>self</th>
<th>cumul</th>
<th>calls</th>
<th>self/call</th>
<th>tot/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.63</td>
<td>2.85</td>
<td>10.08</td>
<td>35.4M</td>
<td>80.5n</td>
<td>80.5n</td>
<td>RanecuEngine::flat()</td>
</tr>
<tr>
<td>3.57</td>
<td>1.81</td>
<td>11.89</td>
<td>3.80M</td>
<td>476n</td>
<td>2.94u</td>
<td>G4Navigator::LocateGlobalPointAndSetup(Hep3Vector const&amp;, Hep3Vector const*, bool, ...)</td>
</tr>
<tr>
<td>3.46</td>
<td>1.75</td>
<td>13.64</td>
<td>5.12M</td>
<td>343n</td>
<td>2.54u</td>
<td>G4SteppingManager::DefinePhysicalStepLength()</td>
</tr>
<tr>
<td>2.29</td>
<td>1.16</td>
<td>24.20</td>
<td>898k</td>
<td>1.30u</td>
<td>7.38u</td>
<td>G4EnergyLoss::GetLossWithFluct(G4DynamicParticle const*, G4Material*, double, ...)</td>
</tr>
<tr>
<td>2.27</td>
<td>1.15</td>
<td>26.35</td>
<td>7.50M</td>
<td>153n</td>
<td>2.27u</td>
<td>G4SteppingManager::InvokePSDIP(unsigned long)</td>
</tr>
<tr>
<td>2.23</td>
<td>1.13</td>
<td>28.48</td>
<td>5.17M</td>
<td>219n</td>
<td>7.53u</td>
<td>G4SteppingManager::Stepping()</td>
</tr>
<tr>
<td>2.17</td>
<td>1.10</td>
<td>30.58</td>
<td>4.92M</td>
<td>223n</td>
<td>2.03u</td>
<td>G4Transportation::PostStepDoIt(G4Track const&amp;, G4Step const&amp;)</td>
</tr>
<tr>
<td>2.12</td>
<td>1.08</td>
<td>32.70</td>
<td>15.8M</td>
<td>67.9n</td>
<td>67.9n</td>
<td>G4PhysicsLogVector::FindBinLocation(double) const</td>
</tr>
<tr>
<td>1.93</td>
<td>0.98</td>
<td>34.63</td>
<td>186n</td>
<td>901n</td>
<td>901n</td>
<td>G4Transportation::AlongStepGetPhysicalInteractionLength(G4Track const&amp;, double, double, double&amp;, G4GPILSelection*)</td>
</tr>
<tr>
<td>1.90</td>
<td>0.96</td>
<td>35.59</td>
<td>22.35</td>
<td>1.11M</td>
<td>864n</td>
<td>864n G4MuPairProduction::ComputeDDMicroscopicCrossSection(G4ParticleDefinition const*, ...)</td>
</tr>
<tr>
<td>1.80</td>
<td>0.91</td>
<td>37.40</td>
<td>23.26</td>
<td>1.01M</td>
<td>897n</td>
<td>1.50u G4MultipleScattering::PostStepDoIt(G4Track const&amp;, G4Step const&amp;)</td>
</tr>
<tr>
<td>1.78</td>
<td>0.90</td>
<td>39.18</td>
<td>24.16</td>
<td>19.1M</td>
<td>47.3n</td>
<td>47.3n G4Track::GetVelocity() const</td>
</tr>
<tr>
<td>1.62</td>
<td>0.82</td>
<td>40.80</td>
<td>24.98</td>
<td>4.48M</td>
<td>182n</td>
<td>720n G4Navigator::ComputeStep(Hep3Vector const&amp;, Hep3Vector const*, double, double&amp;)</td>
</tr>
<tr>
<td>1.51</td>
<td>0.77</td>
<td>42.31</td>
<td>25.75</td>
<td>1.89M</td>
<td>405n</td>
<td>863n G4MultipleScattering::GetContinuousStepLimit(G4Track const&amp;, double, double, double&amp;)</td>
</tr>
<tr>
<td>1.41</td>
<td>0.71</td>
<td>43.72</td>
<td>26.46</td>
<td>4.62M</td>
<td>155n</td>
<td>503n G4ReplicaNavigation::ComputeStep(Hep3Vector const&amp;, Hep3Vector const*, Hep3Vector const&amp;, Hep3Vector const&amp;, double, double&amp;, G4NavigationHistory&amp;, bool&amp;, Hep3Vector&amp;, bool&amp;, bool&amp;, G4VPhysicalVolume**, int&amp;)</td>
</tr>
<tr>
<td>1.27</td>
<td>0.64</td>
<td>45.00</td>
<td>27.10</td>
<td>3.27M</td>
<td>196n</td>
<td>196n G4Tubs::DistanceToOut(Hep3Vector const&amp;, Hep3Vector const&amp;, bool, bool*, Hep3Vector*) const</td>
</tr>
<tr>
<td>1.08</td>
<td>0.55</td>
<td>46.08</td>
<td>27.65</td>
<td>550k</td>
<td>999n</td>
<td>2.28u G4eBremsstrahlung::PostStepDoIt(G4Track const&amp;, G4Step const&amp;)</td>
</tr>
<tr>
<td>1.07</td>
<td>0.54</td>
<td>47.15</td>
<td>28.20</td>
<td>5.01M</td>
<td>109n</td>
<td>170n G4Transportation::AlongStepGetPhysicalInteractionLength(G4Track const&amp;, double, double&amp;)</td>
</tr>
<tr>
<td>0.98</td>
<td>0.50</td>
<td>48.13</td>
<td>28.69</td>
<td>7.00M</td>
<td>70.7n</td>
<td>70.7n _int_malloc</td>
</tr>
<tr>
<td>0.96</td>
<td>0.49</td>
<td>49.12</td>
<td>29.18</td>
<td>4.80M</td>
<td>102n</td>
<td>3.65u G4SteppingManager::InvokePostStepDoItProc()</td>
</tr>
<tr>
<td>0.93</td>
<td>0.47</td>
<td>50.05</td>
<td>29.65</td>
<td>25.8k</td>
<td>18.2u</td>
<td>18.6u G4MultipleScattering::ComputeTransportCrossSection(G4ParticleDefinition const&amp;, ...)</td>
</tr>
<tr>
<td>0.92</td>
<td>0.47</td>
<td>50.97</td>
<td>30.12</td>
<td>4.97M</td>
<td>93.8n</td>
<td>146n Em2SteppingAction::UserSteppingAction(G4Step const*)</td>
</tr>
<tr>
<td>0.91</td>
<td>0.46</td>
<td>51.44</td>
<td>30.58</td>
<td>4.95M</td>
<td>93.4n</td>
<td>1.04u G4SteppingManager::InvokeAlongStepDoItProc()</td>
</tr>
<tr>
<td>0.91</td>
<td>0.46</td>
<td>52.36</td>
<td>31.04</td>
<td>9.64M</td>
<td>47.5n</td>
<td>230n G4VDiscreteProcess::PostStepGetPhysicalInteractionLength(G4Track const&amp;, double, double, double&amp;, G4GPILSelection*)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>G4RandomCondition()</td>
</tr>
</tbody>
</table>
HARDWARE REVIEW
CPU performance vector

- Defined in 3 dimensions

Density of work (cycles)

Instructions (per cycle)

Calculation width (per inst)

Determined by a combination of architecture and microarchitecture
Memory Hierarchy

- From CPU to main memory on Madison
  - With multicore, memory bandwidth is shared between cores on the same bus

CPU (Registers)

L1I (16 KB)
L1D (16 KB)

L2 (256 KB)

L3 (9 MB)

memory (large)

32B/c, 5 - 7 c latency
32B/c, 12 - 15 c latency

~4 B/c, ~200 c. latency
Cache lines

• Madison L3 cache lines are 128B (16 * double)
  – Minimum amount of data transferred between cache and memory.
  – Imagine what happens if your stride is 16 (or more)!

Programming the memory hierarchy is an art in itself.
Back to Compilers
## TestKalman [nx,ny] : kalman_win7.1

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.61</td>
<td>0.55</td>
<td>0.60</td>
<td>1.26</td>
<td>2.16</td>
<td>3.91</td>
<td>5.79</td>
<td>7.43</td>
<td>9.66</td>
</tr>
<tr>
<td>2</td>
<td>0.90</td>
<td>0.64</td>
<td>0.96</td>
<td>1.63</td>
<td>2.15</td>
<td>3.16</td>
<td>4.05</td>
<td>4.93</td>
<td>5.81</td>
</tr>
<tr>
<td>3</td>
<td>0.57</td>
<td>0.70</td>
<td>0.98</td>
<td>1.40</td>
<td>1.83</td>
<td>2.60</td>
<td>3.67</td>
<td>4.68</td>
<td>5.69</td>
</tr>
<tr>
<td>4</td>
<td>0.34</td>
<td>0.50</td>
<td>1.07</td>
<td>1.53</td>
<td>1.88</td>
<td>2.50</td>
<td>3.16</td>
<td>3.82</td>
<td>4.55</td>
</tr>
<tr>
<td>5</td>
<td>0.83</td>
<td>0.85</td>
<td>0.54</td>
<td>1.52</td>
<td>1.91</td>
<td>2.38</td>
<td>3.16</td>
<td>3.82</td>
<td>4.55</td>
</tr>
<tr>
<td>6</td>
<td>1.31</td>
<td>0.84</td>
<td>1.05</td>
<td>1.08</td>
<td>1.43</td>
<td>1.76</td>
<td>2.10</td>
<td>2.43</td>
<td>2.76</td>
</tr>
<tr>
<td>7</td>
<td>0.85</td>
<td>1.16</td>
<td>1.68</td>
<td>2.25</td>
<td>3.05</td>
<td>5.02</td>
<td>7.11</td>
<td>9.15</td>
<td>11.20</td>
</tr>
<tr>
<td>8</td>
<td>0.96</td>
<td>1.08</td>
<td>1.69</td>
<td>2.23</td>
<td>3.05</td>
<td>5.02</td>
<td>7.11</td>
<td>9.15</td>
<td>11.20</td>
</tr>
</tbody>
</table>

N1,N2 <= 6: 36.51 37.96 66.81
N1,N2 > 6: 281.15 242.13 537.41 540.08 485.39

**SMatrix_Sym** **SMatrix** **TMatrix** **SMatrix_Sym better than TMatrix**

## TestKalman [nx,ny] : kalman_solaris.5.9

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.29</td>
<td>4.53</td>
<td>7.49</td>
<td>13.38</td>
<td>27.52</td>
<td>36.68</td>
<td>42.12</td>
<td>56.27</td>
<td>74.79</td>
</tr>
<tr>
<td>2</td>
<td>1.29</td>
<td>2.41</td>
<td>3.52</td>
<td>5.67</td>
<td>8.68</td>
<td>20.13</td>
<td>29.90</td>
<td>41.39</td>
<td>56.98</td>
</tr>
<tr>
<td>3</td>
<td>2.40</td>
<td>2.95</td>
<td>3.84</td>
<td>5.19</td>
<td>8.18</td>
<td>14.74</td>
<td>25.32</td>
<td>42.51</td>
<td>51.08</td>
</tr>
<tr>
<td>4</td>
<td>3.36</td>
<td>6.28</td>
<td>9.88</td>
<td>17.60</td>
<td>33.32</td>
<td>37.52</td>
<td>51.27</td>
<td>69.29</td>
<td>86.88</td>
</tr>
<tr>
<td>5</td>
<td>2.07</td>
<td>3.43</td>
<td>5.99</td>
<td>7.79</td>
<td>12.81</td>
<td>24.26</td>
<td>35.10</td>
<td>59.11</td>
<td>66.99</td>
</tr>
<tr>
<td>6</td>
<td>0.83</td>
<td>1.71</td>
<td>1.98</td>
<td>1.75</td>
<td>2.63</td>
<td>4.27</td>
<td>6.08</td>
<td>8.90</td>
<td>11.71</td>
</tr>
<tr>
<td>7</td>
<td>0.80</td>
<td>1.67</td>
<td>1.94</td>
<td>1.71</td>
<td>2.60</td>
<td>4.22</td>
<td>6.02</td>
<td>8.84</td>
<td>11.65</td>
</tr>
<tr>
<td>8</td>
<td>0.78</td>
<td>1.64</td>
<td>1.91</td>
<td>1.68</td>
<td>2.56</td>
<td>4.18</td>
<td>6.00</td>
<td>8.81</td>
<td>11.62</td>
</tr>
</tbody>
</table>

N1,N2 <= 6: 445.36 218.23 264.08 1873.16 1837.24
N1,N2 > 6: 3995.72 2999.65 3541.10

**SMatrix_Sym** **SMatrix** **TMatrix** **SMatrix_Sym better than TMatrix**

---

## TestKalman [nx,ny] : kalman_slc3_gcc323

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.26</td>
<td>0.33</td>
<td>0.38</td>
<td>0.59</td>
<td>0.86</td>
<td>1.54</td>
<td>2.07</td>
<td>2.69</td>
<td>3.30</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>0.34</td>
<td>0.38</td>
<td>0.59</td>
<td>0.86</td>
<td>1.54</td>
<td>2.07</td>
<td>2.69</td>
<td>3.30</td>
</tr>
<tr>
<td>3</td>
<td>0.46</td>
<td>0.50</td>
<td>0.54</td>
<td>0.75</td>
<td>0.98</td>
<td>1.64</td>
<td>2.32</td>
<td>3.00</td>
<td>3.68</td>
</tr>
<tr>
<td>4</td>
<td>0.47</td>
<td>0.50</td>
<td>0.54</td>
<td>0.75</td>
<td>0.98</td>
<td>1.64</td>
<td>2.32</td>
<td>3.00</td>
<td>3.68</td>
</tr>
<tr>
<td>5</td>
<td>0.46</td>
<td>0.50</td>
<td>0.54</td>
<td>0.75</td>
<td>0.98</td>
<td>1.64</td>
<td>2.32</td>
<td>3.00</td>
<td>3.68</td>
</tr>
<tr>
<td>6</td>
<td>0.46</td>
<td>0.50</td>
<td>0.54</td>
<td>0.75</td>
<td>0.98</td>
<td>1.64</td>
<td>2.32</td>
<td>3.00</td>
<td>3.68</td>
</tr>
<tr>
<td>7</td>
<td>0.46</td>
<td>0.50</td>
<td>0.54</td>
<td>0.75</td>
<td>0.98</td>
<td>1.64</td>
<td>2.32</td>
<td>3.00</td>
<td>3.68</td>
</tr>
<tr>
<td>8</td>
<td>0.46</td>
<td>0.50</td>
<td>0.54</td>
<td>0.75</td>
<td>0.98</td>
<td>1.64</td>
<td>2.32</td>
<td>3.00</td>
<td>3.68</td>
</tr>
</tbody>
</table>

N1,N2 <= 6: 29.45 32.23 34.08 36.35 36.53
N1,N2 > 6: 264.57 265.67 266.52 267.27 268.08

**SMatrix_Sym** **SMatrix** **TMatrix** **SMatrix_Sym better than TMatrix**
“Low-hanging fruit”

• Typically one starts with a given compiler, and moves to:

• More aggressive compiler options
  – For instance:
  – -O2 → -O3, -funroll-loops, -ffast-math (g++)
  – -O2 → -O3, -ipo (icc)

• More recent compiler versions
  – g++ version 3 → g++ version 4
  – icc version 8 → icc version 9

• Different compilers
  – GNU → Intel (or reverse?)

Some options can compromise accuracy or correctness

May be a burden because of potential source code issues
Interprocedural optimization

• Let the compiler worry about interprocedural relationship
  – “icc –ipo”
• Valid also when building libraries
  – Archive
  – Shared
• Cons:
  – Can lead to code bloat
  – Longer compile times

Probably most useful when combined with heavy optimization for “production” binaries or libraries!
Feedback Optimization

• Many compilers allow further optimization through training runs
  – Compile once (to instrument binary)
    • g++ -fprofile-generate
    • icc -prof_gen
  – Run one (or several test cases)
    • ./test40 < test40.in (will run slowly)
  – Recompile w/feedback
    • g++ -fprofile-use
    • icc -prof_use (best results when combined with -O3,-ipo

With icc 9.0 we get ~20% on root stress tests on Itanium, but only ~5% on x86-64
CONCLUSION
Conclusions

- Understand which parts of the “spiral” you control
- Understand the platform hardware
- Equip yourself with good tools
  - Get access to hw performance counters
  - Exploit the power of performance tools
- Check how key algorithms map on to your hardware platform
  - Are you at 5% or 95% efficiency?
  - Where do you want to be?
- Cycle around the spiral frequently
  - It is hard to get to “peak” performance (and stay there!)
QUESTIONS?
Backup
In comes the PMU (Performance Monitoring Unit)

Quickly summarized:
4 counters (12 on Montecito)
~200 monitored events
Some very advanced features!
Itanium-2 cache hierarchy
Geant 4 – Test40

- Overall counters in $10^9$

<table>
<thead>
<tr>
<th>Counter</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA64_INST_RETIRED</td>
<td>108.43</td>
</tr>
<tr>
<td>NOPS_RETIRED</td>
<td>39.73</td>
</tr>
<tr>
<td>CPU_CYCLES (CC)</td>
<td>74.98</td>
</tr>
<tr>
<td>BACK_END_BUBBLE_ALL</td>
<td>43.10</td>
</tr>
<tr>
<td>Useful instructions (UI)</td>
<td>68.70</td>
</tr>
<tr>
<td>Non-stalled cycles (NSC)</td>
<td>31.88</td>
</tr>
<tr>
<td>UI/CC</td>
<td>~ 1</td>
</tr>
<tr>
<td>UI/NSC</td>
<td>~ 2</td>
</tr>
</tbody>
</table>
• **Stall counters**

<table>
<thead>
<tr>
<th>Counter</th>
<th>Counts (10^9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACK_END_BUBBLE_ALL</td>
<td>43.10</td>
</tr>
<tr>
<td>BE_EXE_BUBBLE_ALL</td>
<td>27.96</td>
</tr>
<tr>
<td>BACK_END_BUBBLE_FE</td>
<td>7.38</td>
</tr>
<tr>
<td>BE_L1D_FPU_BUBBLE_ALL</td>
<td>3.86</td>
</tr>
<tr>
<td>BE_FLUSH_BUBBLE_ALL</td>
<td>2.72</td>
</tr>
<tr>
<td>BE_RSE_BUBBLE_ALL</td>
<td>1.38</td>
</tr>
</tbody>
</table>

**Why so many EXE bubbles?**

**Keep drilling down!**

65%
### Geant 4 – Test40

#### EXE stall counters

<table>
<thead>
<tr>
<th>Counter</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>BE_EXE_BUBBLE_ALL</td>
<td>27.96</td>
</tr>
<tr>
<td>BE_BUBBLE_GRALL</td>
<td>10.29</td>
</tr>
<tr>
<td>BE_BUBBLE_GRGR</td>
<td>~zero</td>
</tr>
<tr>
<td>BE_EXE_BUBBLE_FRALL</td>
<td>17.21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Counter</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2_REFERENCES</td>
<td>19.39</td>
</tr>
<tr>
<td>L2_DATA_REFERENCES_L2_ALL</td>
<td>13.16</td>
</tr>
</tbody>
</table>
Test40 - Cache counters
Software Pipelining
Mersenne Twister

```c
Double_t TRandom3::Rndm(Int_t) {
    Uint_t y;
    const Int_t kM = 397; const Int_t kN = 624; const Uint_t kTemperingMaskB = 0x9
    const Uint_t kTemperingMaskC = 0xefc60000; const Uint_t kUpperMask = 0x800
    const Uint_t kLowerMask = 0x7fffffff; const Uint_t kMatrixA = 0x990
    if (fCount624 >= kN) {
        register Int_t i;
        for (i=0; i < kN-kM; i++) { /* THE LOOPS */
            y = (fMt[i] & kUpperMask) | (fMt[i+1] & kLowerMask);
            fMt[i] = fMt[i+kM] ^ (y >> 1) ^ ((y & 0x1) ? kMatrixA : 0x0);
        }
        for ( ; i < kN-1 ; i++) {
            y = (fMt[i] & kUpperMask) | (fMt[i+1] & kLowerMask);
            fMt[i] = fMt[i+kM-kN] ^ (y >> 1) ^ ((y & 0x1) ? kMatrixA : 0x0);
        }
        y = (fMt[kN-1] & kUpperMask) | (fMt[0] & kLowerMask);
        fMt[kN-1] = fMt[kM-1] ^ (y >> 1) ^ ((y & 0x1) ? kMatrixA : 0x0);
        fCount624 = 0;
    }
    y = fMt[fCount624++]; /*THE STRAIGHT-LINE PART*/
    y ^= (y >> 11); y ^= ((y << 7 ) & kTemperingMaskB );
    y ^= ((y << 15) & kTemperingMaskC ); y ^= (y >> 18);
    if (y) return ((Double_t) y * 2.3283064365386963e-10); // * Power(2,-32)
    return Rndm();
}
```
The “MT” loop is full

- **Highly optimized**
  - Here depicted in 3 Itanium cycles
  - But similarly dense on other platforms

<table>
<thead>
<tr>
<th></th>
<th>Load</th>
<th>Test Bit</th>
<th>XOR</th>
<th>Load</th>
<th>Add</th>
<th>No-op</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>AND</td>
<td>AND</td>
<td>Shift</td>
<td>Add</td>
<td>Load</td>
<td>Move</td>
</tr>
<tr>
<td>2</td>
<td>Store</td>
<td>OR</td>
<td>XOR</td>
<td>Add</td>
<td>Add</td>
<td>Branch</td>
</tr>
</tbody>
</table>
The sequential part is not!

```plaintext
y = fMt[fCount624++] ; /*THE STRAIGHT-LINE PART*/
    y ^=  (y >> 11);  y ^= ((y << 7 ) & kTemperingMaskB );
    y ^= ((y << 15) & kTemperingMaskC );  y ^=  (y >> 18);
if (y) return ( (Double_t) y * 2.3283064365386963e-10);
```

<table>
<thead>
<tr>
<th>No.</th>
<th>Opcode 1</th>
<th>Opcode 2</th>
<th>Opcode 3</th>
<th>Opcode 4</th>
<th>Opcode 5</th>
<th>Opcode 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Add</td>
<td>Mov long</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>1</td>
<td>Load</td>
<td>Mov long</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>2</td>
<td>Shift,11</td>
<td>Set float</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>3</td>
<td>XOR</td>
<td>Move</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>4</td>
<td>Shift,7</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>5</td>
<td>AND</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>6</td>
<td>XOR</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>7</td>
<td>SHL,15</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>8</td>
<td>AND</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>9</td>
<td>XOR</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>10</td>
<td>SHL,18</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>11</td>
<td>XOR</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>12</td>
<td>Set float</td>
<td>Compare</td>
<td>Branch</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>13</td>
<td>Bubble (no work dispatched, because of FP latency)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Bubble (no work dispatched, because of FP latency)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Bubble (no work dispatched, because of FP latency)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Bubble (no work dispatched, because of FP latency)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Bubble (no work dispatched, because of FP latency)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Mult FP</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
<tr>
<td>19</td>
<td>Bubble (no work dispatched, because of FP latency)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Bubble (no work dispatched, because of FP latency)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Bubble (no work dispatched, because of FP latency)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Mult FP</td>
<td>Branch</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
<td>No-op</td>
</tr>
</tbody>
</table>
```