Application Performance and the ToolBox
Jim Gochee, PowerPC
1/30/93
Rev 3

Overview

The true performance of any machine is measured by how fast its applications run, not necessarily how fast the CPU is. PowerPC is no exception. While the 601 is 3 to 4 times faster than a 25mhz 68040 (integer specmarks), applications that are emulated, or applications that call an emulated ToolBox, may see only a fraction of this horsepower. On average, native code executes 10 times faster than emulated code, which means an emulated application calling an emulated ToolBox would be 10 times slower than a fully native machine. Because the first PowerPC release will have significant amounts of emulated code, it is important to understand how fast applications can be expected to run in a mixed code environment. By varying the ratio of emulated instructions to native instructions, we can get a feeling for the machine's overall performance.

Applications that execute 30% emulated code and 70% native code will be 3.7 times slower than a fully native machine, or at the speed of a 25mhz 68040. However, as the amount of emulated code approaches 0%, Amdahl's law points out that gains will be offset by the high cost of executing emulated code. For instance, even if an application executes 90% native code, it will only be 2 times as fast as a 25mhz 68040.

The Facts

This table shows the effect of the equation:

NATIVE + (10)EMULATED = AVG_NATIVE_INSTRUCTIONS

NATIVE is the percentage of time spent executing native code, EMULATED is the percentage of time spent executing emulated code, and AVG_NATIVE_INSTRUCTIONS is the average number of native instructions per mixed instruction type. EMULATED is multiplied by 10 because each emulated instruction takes around 10 cycles, whereas each native instruction takes one cycle. At 30% emulated code, the machine should perform as well as a 25mhz 68040.

% Emulated Code	Avg Native Instructions
0	1
1	1.09
2	1.18
3	1.27
4	1.36

5	1.45
10	1.9
15	2.35
20	2.8
25	3.25
30	3.7
35	4.15
40	4.6
45	5.05
50	5.5
55	5.95
60	6.4
65	6.85
70	7.3
75	7.75
80	8.2
85	8.65
90	9.1
95	9.55
100	10

Emulated Application Performance

68k applications will probably not reach 25mhz 68040 performance by V0, though they will come close. Studies show that applications generally spend 60-80% of their time in the ToolBox. An average application will execute its own 68k code (30%) plus some amount of 68k ToolBox code. Given two scenarios, one where 70% of the executed ToolBox instructions are native, and another where 90% are native, the average emulated application will execute at 74% of a 25mhz 68040.

$$(10 * .30) + ((1 * .49) + (10 * .21)) = 5.59$$

$$(10 * .30) + ((1 * .63) + (10 * .07)) = 4.33$$

Average = 4.96, compared with 3.7 for a 25mhz 68040, (3.7/4.96) = 74%

Native Application Performance

For native applications to fully realize the potential of the PowerPC, it is imperative that there is as little 68k code as possible. This means that almost all ToolBox routines (and the routines they in turn call) should be native. Since this is unrealistic for V0, there will be a portion of the ToolBox that is native and the rest will be 68k. To see how quickly the performance of a native application can deteriorate as the percentage of 68k code increases, let's assume that applications execute 70% of their instructions in the ToolBox. The worst case scenario is that the entire ToolBox is emulated, which from the equation below shows the machine running 7.3 times slower than it would if no 68k code were executed. Another way to look at it is that the native application would be 2 times slower than a 25mhz 68040. This is the worst case.

$$(1 * .30) + (10 * .70) = 7.3$$

Now suppose that enough of the ToolBox is ported so that 80% of the instructions executed in the ToolBox are native. We still find that the machine is more than twice as slow as its potential and only 61% faster than a 25mhz 68040.

$$(1 * .30) + ((1 * .56) + (10 * .14)) = 2.26$$

Just in case you think Amdahl's law is horse radish, assume that 95% of the instructions executed in the ToolBox are native. We still see that the machine could go approximately 1/3 faster if all the ToolBox was native.

$$(1 * .30) + ((1 * .665) + (10 * .035)) = 1.315$$

Context Switches

In the previous calculations, it was assumed that some fraction of the instructions were 68k and some 601. What was left out was the overhead of the machine switching back and forth between the two code types. This switch incurs a penalty of 10-14µ, or 500-750 native instructions! Not only is it important to keep the amount of emulated code to a minimum, but it is crucial to limit the number of switches. For this reason, ToolBox call chains must be completely ported.

The Plan

Based on the assumption that we have a limited time to port ToolBox routines, it is imperative that we focus our energy wisely. The first step is to fully understand how applications interact with the ToolBox. By examining performance sensitive sections of native applications with the PowerProfiler, we can pinpoint routines to port. Also, the PowerProfiler will enable us to track context switches so that we can be 100% guaranteed that our execution paths stay native. Hopefully, we will have areas of the ToolBox where 100% of the code is native, and other areas where 0% of the code will be native. Decisions will have to be made about which areas of performance are most critical to the user.

While it appears that 68k applications should perform well compared to a 25mhz 68040, native applications will be severely limited unless they access a native ToolBox. We will spend much of our energy analyzing the Inside Track applications in search of common patterns and behaviors. With luck and lots of porting we hope to accelerate speed sensitive portions of these applications and give a sense for what the future of RISC will look like.

Current Status

We are currently focused on porting the ToolBox in the areas of Quickdraw and text display. All high level APIs are to be ported along with complete call chains. Graphical output is usually the most speed sensitive part of the ToolBox, so we hope to get big wins from this. The PowerProfiler is being used on native applications created by MSD and MSSW to determine what to port next. Our next step is to get Inside Track applications as they are ported native. Guidance from Pierre Cesarini and Jordan Mattson will help keep us aware of weak spots in the system and areas where performance is critical. Jordan will coordinate with developers to get feedback as concerning where they think their applications should perform well.