

# PowerPC / IRM Task Timing

## *Early performance comparison*

Tue, Apr 17, 2001

The front-ends used in many projects at Fermilab are IRMs that are based on the 68040 CPU. A new version of the system code is based upon the PowerPC and will be used as an upgrade to the earlier 68020-based front ends used to support Linac controls. The PowerPC is expected to leap frog over the IRM performance level. This note compares some of early task activity timing between the new PowerPC-based systems and the 68040-based IRMs.

Whenever timing comparisons are made, one has to take all potential variations into account. The IRMs run without "little console" hardware, for example. One PowerPC system used for timing measurements was connected to such console hardware. This is significant because it takes more time to write to the real console video ram than to a virtual display, since we throttle writes to video ram to the raster scan retrace time to keep from causing "snow" on the screen during updates. Until we improve this throttling logic, we are limited to a 15  $\mu$ s/character average writing time to the screen. Writing 400 characters to the screen can therefore take 6 ms, independent of the power of the CPU. But it's a bit more complicated than that. The raster scan timing is obtained by monitoring a status bit whose state identifies the acceptable time to write to the video ram. When this status bit indicates ok, up to 4 characters are written. But if the next 4 characters are ready while the status bit is still ok, they can piggy-back on the same raster scan time, which is about 64  $\mu$ s. Thus, it does depend upon the CPU performance.

### *Task timing results*

Extracted from recent task timing examples, here are some comparisons:

<i>Task</i>	<i>PowerPC</i>	<i>IRM</i>	<i>Activity</i>
Cons	.01	.07	Usual Console Task idle
DTim	.01	.08	Usual time-of-day marking
QMon	.01	.06	Queue monitor idle
Appl	.02	.12	Page A idle
Serv	.01	.07	Server Task--no server replies due

With the times quoted in ms, the above few examples measure very short activities, and therefore not very important ones. But they show that the PowerPC CPU is often 6–8 times as fast as the 68040. (The 68K system code is written in assembly language, whereas the PowerPC code is written in C. From previous study, we regard this language factor as a 30% effect, which is mostly ignored here.)

<i>Task</i>	<i>PowerPC</i>	<i>IRM</i>	<i>Activity</i>
Clas	.08	.75	Request for 64-byte analog descriptor
Updt	.14	.34	Build reply to above request
tSX	.09	—	Time to transmit reply datagram

In this simple example of a one-shot request/reply, interpretation of the request shows the full PowerPC enhanced performance. But building the reply message is not quite so efficient for the PowerPC. It may be that the network stack works harder. The tSX task is the Transmitter Task that invokes `sendto()`; it does not return until the transmission is finished. By comparison, the IRM software buffers the datagram to the network hardware, allowing further processing to continue. Still, in this example, the PowerPC is still an improvement over the IRM.

<i>Task</i>	<i>PowerPC</i>	<i>IRM</i>	<i>Activity</i>
Appl	.45	2.49	Page A keyboard int: request descriptor
Appl	1.60	1.17	Page A write reply data to screen

In processing the keyboard interrupt action to build the request, the PowerPC is again 5 times faster. When updating the screen with the reply data, however, we must take into account the relatively slow logic that writes to the screen, as detailed above. If the IRM were connected to a little console, it would have been much slower.

<i>Task</i>	<i>PowerPC</i>	<i>IRM</i>	<i>Activity</i>
ALRM	1.94	1.15	Alarm scan for 1024 channels, 1024 bits

The alarm scan timing here really measures the overhead in scanning so many entries in the appropriate tables, since nearly all entries in these nodes are inactive, or "bypassed." But IRMs with many channels enabled for checking do not vary much from this. The reason why the IRM shows a faster scan time is that this alarm scan logic is heavily dominated by accesses to the nonvolatile memory tables, especially the ADATA table. In the PowerPC system, this nonvolatile memory has a very slow access, on the order of 1  $\mu$ s. Furthermore, such accesses cannot be cached, lest the nonvolatility of the memory be found lacking. (We learned this empirically.) So the PowerPC timing takes a severe "hit" in this case. Still, this job only has to be performed every 66 ms, so there is time available. If we double the number of channels and bits, this time will also double. It is anticipated that some of the upgraded Linac nodes will be doubled.

#### *Acnet-related times*

In an Acnet front-end, concern may be raised about the timing of Acnet communications, especially RETDAT requests and replies. When many requests from Acnet consoles are active, especially in the 15 Hz Linac nodes, the front-ends bear a heavy burden in having to fulfill many requests every cycle.

One example is the time to process a request for 50 analog 2-byte readings. This is a real-life case for Linac, in that a commonly-used display program that plots beam toroids and loss monitors throughout the Linac is used for this very purpose. The request message is about 800 bytes. The reply message is about 200 bytes, since each reading is accompanied by a status word. Here are results for the front-end receiving such a request and replying to it.

<i>Tasks</i>	<i>PowerPC</i>	<i>IRM</i>	<i>133</i>	<i>Activity</i>
Request	.33	2.12	7.0	Receive, initialize request
Reply	.26	.88	2.0	Build reply, transmit

For the PowerPC case, the request time includes the time needed for the Net, tSRd, ANET, and ACRQ tasks. The reply timing includes the timing for both the Updt task that builds the reply datagram and the tSX task that invokes sendto() to transmit it. For the IRM and 133 (MVME-133 board with 68020 CPU as used in the present Linac nodes) cases, the request timing includes the SNAP, ANET and ACRQ tasks; the reply time includes only the Updt task time, since the actual time to transmit overlaps subsequent task processing. It is obvious from these results that the PowerPC performance will be a giant leap forward compared to that of the present Linac front-ends. Replies will be able to be delivered to many more Acnet consoles simultaneously.

#### *Data Access Table timing*

Every 15 Hz cycle, the first job for all these front-ends is to update the data pool by following "instructions" in the Data Access Table, following by updating all active non-server data requests with replies that are due on that cycle. A built-in diagnostic measures the elapsed time for each instruction entry in this table. Here are some representative samplings of this data:

<i>Instruction</i>	<i>PowerPC</i>	<i>IRM</i>	<i>133</i>	<i>Activity</i>
RBinary	.40	.15	1.5	Update digital data pool

In the RBinary case, in which all binary data bytes are read to update the digital data pool, some explanation is required. All addresses of digital data bytes are initialized to point to nonvolatile

memory by default, which implies they are "software" digital bytes. Then hardware addresses are filled in as needed. Because most of them in these examples are therefore pointing to nonvolatile memory bytes, the slow accesses to such memory in PowerPC systems dominate the performance. Even so, there is an improvement over the present Linac nodes. But another wrinkle will make the PowerPC results worse in this case. For the klystron front-ends, many digital bytes are accessed via the Vertical Interconnect scheme, and such accesses to other VME crate memory cost  $4 \mu\text{s}$  each. Most klystron stations have about 40 such bytes of digital data, so this part alone will take about 0.16 ms. In Linac stations, another detail comes into play, as the digital data includes many bytes that come via part of the 15 Hz arcnet transmission from the SRMs. It's a bit difficult to place a number on it, but the upgraded systems will have fewer accesses to nonvolatile memory bytes as a result. Out of a total of 128 bytes digital data bytes, a typical klystron node may have 50 bytes coming from SRMs. Any way one analyzes it, there will be an improvement in the upgraded Linac nodes.

<i>Instruction</i>	<i>PowerPC</i>	<i>IRM</i>	<i>133</i>	<i>Activity</i>
SRMwait	10	10	10	Await data from one or more SRMs

This result merely indicates that the waiting time for SRM data does not depend upon the CPU's performance. Only the response time of the 68332 CPU in the SRM hardware matters. One could ask why we cannot overlap the waiting time with some more useful activity. But there is not really any useful activity that can be done at this time early in the 15 Hz cycle, since we are trying to maintain a facade of a data pool that updates instantaneously from the point-of-view of an outside user. We cannot perform an alarm scan, nor execute the local applications, since the data pool is not yet ready with fresh data. We cannot entertain data requests, because they will likely try to access the data pool. There is no significant work that can be done, so the Update Task waits for the SRM replies, or it times out if they are not forthcoming. This is one part that the PowerPC cannot improve upon in the Linac upgrade.

The time required to execute local applications each cycle is harder to pin down. But the PowerPC's basic CPU performance is expected to make a big difference here. This will especially be true for floating point calculations, even though the Linac nodes currently include the 68881 floating point processor. The PowerPC 750 chip built-in floating point is very much faster.

All in all, the PowerPC nodes, even after waiting for SRM data, are expected to support much greater loading than the current Linac front-ends. Also, the ethernet interface hardware is much faster than the current token ring chip support.