Sqrt() Incidence: Sep. 18, 2019

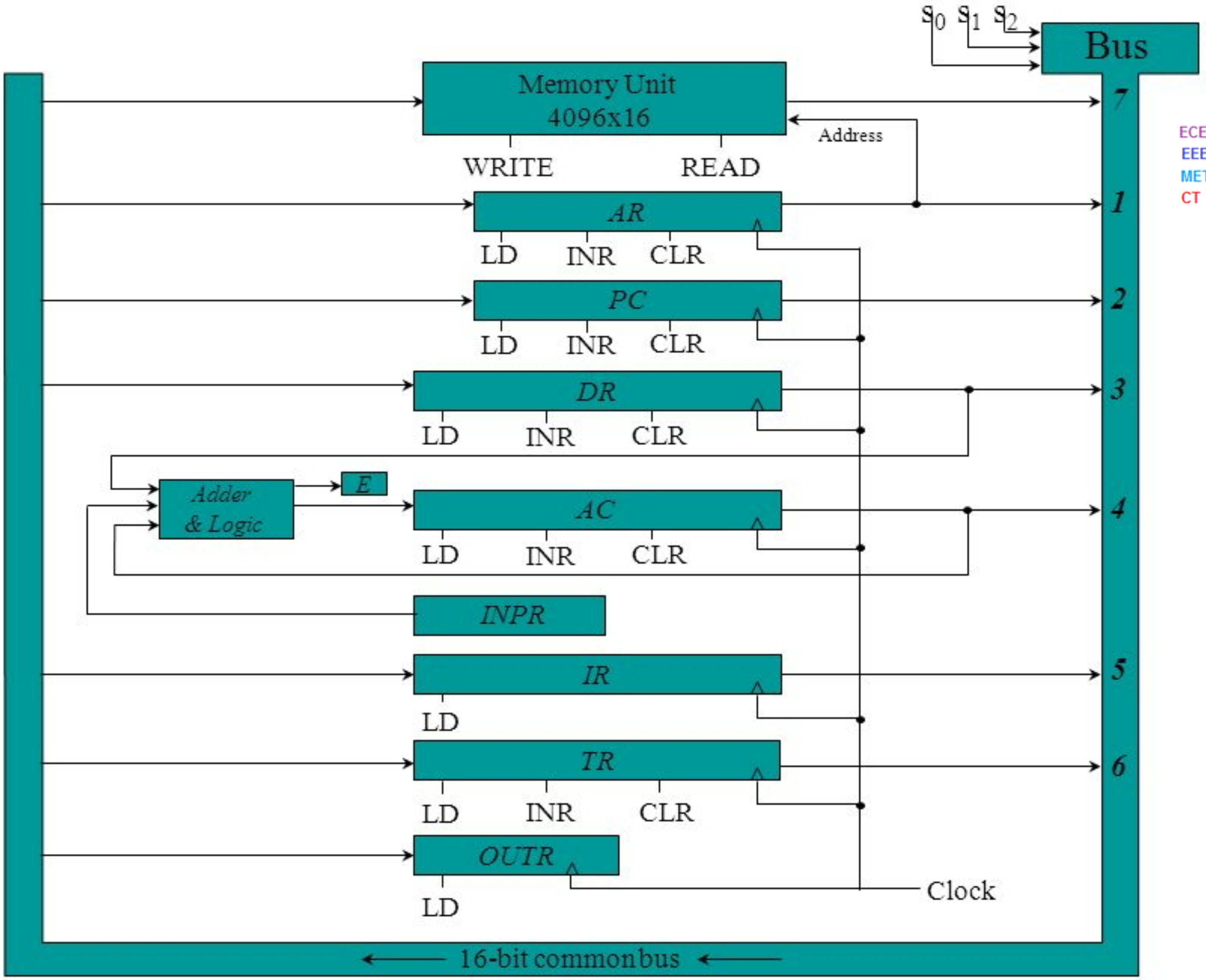
From: "Luo, Ye" <yeluo@anl.gov>
Subject: Re: LED
Date: September 18, 2019 at 2:41:16 PM PDT
To: Aiichiro Nakano <anakano@usc.edu>, Ken-Ichi Nomura <knomura@usc.edu>
Cc: Pankaj Rajak <rajak@usc.edu>

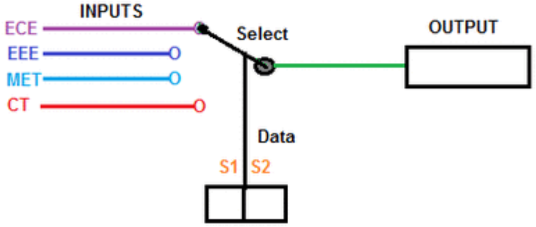
We have no luck with GCC for offload sqrt() as well. I'm giving high hopes to Intel delivering the beta compiler release in two weeks. Intel fortran side is lagging behind and none of the kernels Pankaj and I made works so far. Once we verified the situation with the beta release, we will move or file bug reports.

Best,
Ye

```
> From: Aiichiro Nakano <anakano@usc.edu>
> Sent: Wednesday, September 18, 2019 4:37 PM
> To: Ken-Ichi Nomura <knomura@usc.edu>
> Cc: Luo, Ye <yeluo@anl.gov>; Pankaj Rajak <rajak@usc.edu>
> Subject: Re: LED
>
> Thank you very much, Kenichi. Something like this for now?
>
> On Sep 18, 2019, at 2:24 PM, Ken-Ichi Nomura <knomura@usc.edu> wrote:
>
> Dear All,
>
> Summary of today’s work. Code compilation was successful by commenting out sqrt(). We would need to use a hand-written sqrt() for now.
>
> # env vars for profiling
> export LIBOMPTARGET_PROFILE=T,usec
> export LIBOMPTARGET_DEBUG=1
>
> # Intel compiler + IRIS GPU test
> qsub -I -q iris -t 30 -n 1
> export MODULEPATH=/soft/restricted/CNDA/modulefiles
> module load omp
> icpx -fiopenmp -fopenmp-targets=spir64=-fno-exceptions led.C four1s.c -o led
>
> # GNU compiler + NVIDIA GPU test
> qsub -I -q gpu_mules -t 30 -n 1
> export MODULEPATH=$MODULEPATH:/soft/restricted/intel_dga/modulefiles:/home/yeluo/privatemodules
> module load openmpi/2.1.6-gcc gcc/9.2
> /soft/libraries/mpi/openmpi/2.1.6/bin/mpic++ -fopenmp -foffload=nvptx-none -foffload=-lm led.C four1s.c -o led
```

Basic Computer Architecture





Computer System Architecture, Mano, Copyright (C) 1993 Prentice-Hall, Inc.

M. M. Mano, *Computer System Architecture* (Prentice-Hall)

FLOATING-POINT UNIT DESIGN

USING TAYLOR-SERIES EXPANSION ALGORITHMS

by

Taek-Jun Kwon

Thesis Proposal

UNIVERISITY OF SOUTHERN CALIFORNIA
ELECTRICAL ENGINEERING

September 2006

How Time Consuming Is SQRT()?

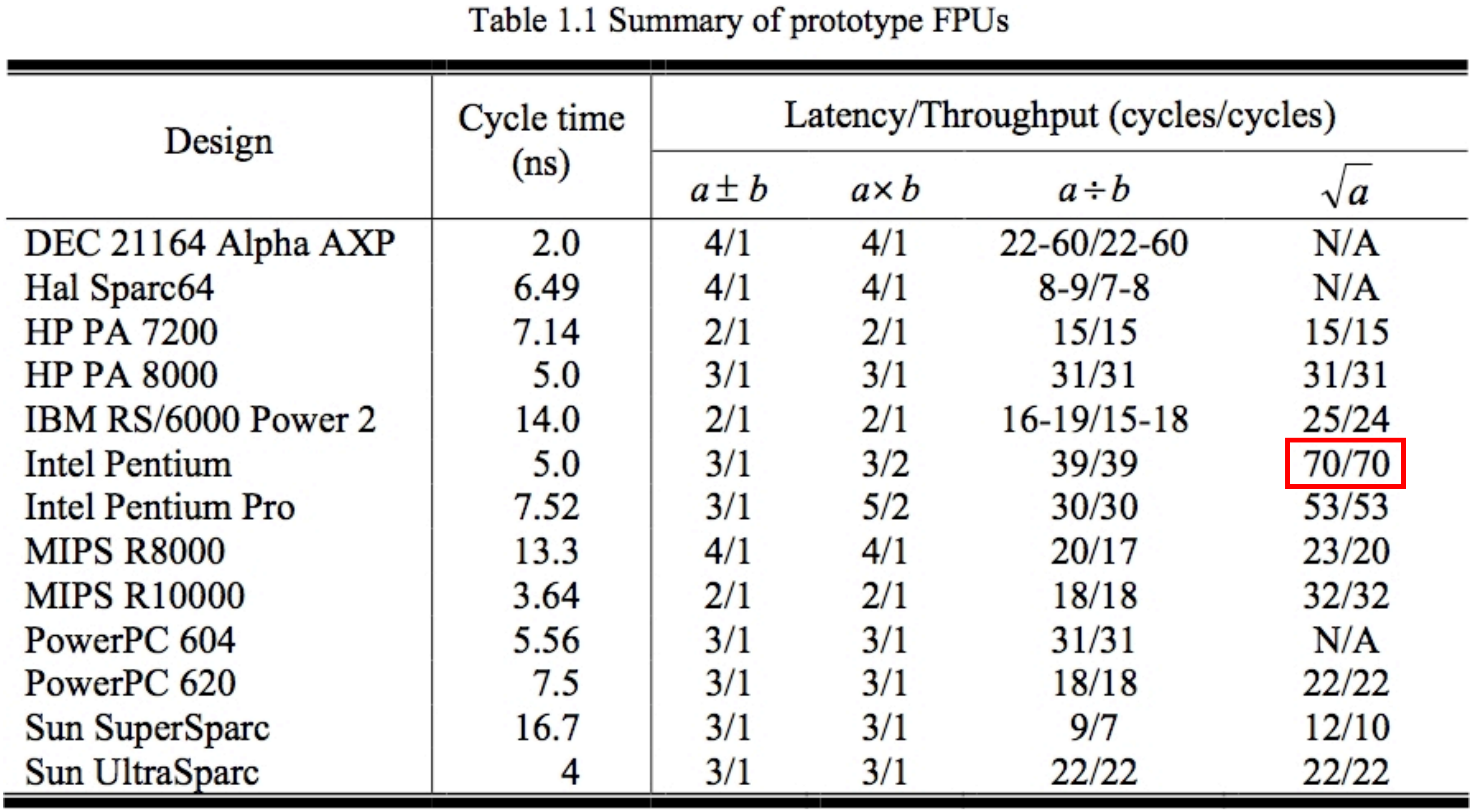


Table 1.1 Summary of prototype FPUs

Design	Cycle time (ns)	Latency/Throughput (cycles/cycles)			
		$a \pm b$	$a \times b$	$a \div b$	$\sqrt{a}$
DEC 21164 Alpha AXP	2.0	4/1	4/1	22-60/22-60	N/A
Hal Sparc64	6.49	4/1	4/1	8-9/7-8	N/A
HP PA 7200	7.14	2/1	2/1	15/15	15/15
HP PA 8000	5.0	3/1	3/1	31/31	31/31
IBM RS/6000 Power 2	14.0	2/1	2/1	16-19/15-18	25/24
Intel Pentium	5.0	3/1	3/2	39/39	70/70
Intel Pentium Pro	7.52	3/1	5/2	30/30	53/53
MIPS R8000	13.3	4/1	4/1	20/17	23/20
MIPS R10000	3.64	2/1	2/1	18/18	32/32
PowerPC 604	5.56	3/1	3/1	31/31	N/A
PowerPC 620	7.5	3/1	3/1	18/18	22/22
Sun SuperSparc	16.7	3/1	3/1	9/7	12/10
Sun UltraSparc	4	3/1	3/1	22/22	22/22

- **Latency: How many clock cycles to compete 1 operation**
- **Throughput: Cycles before the next operation can be issued**

Hardware Implementation of SQRT()

- **Newton-Raphson method**

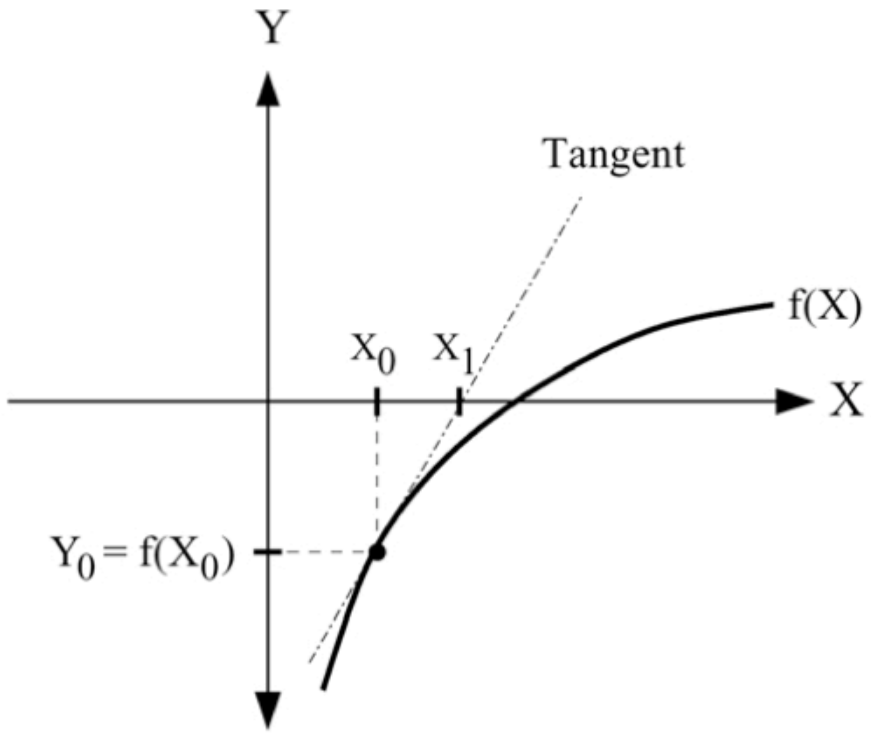


Figure 2.1 Newton-Raphson algorithm for finding the root of *f(x)*

- **Series expansion**

$$\sqrt{b} \approx Y_0\left\{1-\frac{1}{2}\left(1-\frac{b}{Y_0{}^2}\right)-\frac{1}{8}\left(1-\frac{b}{Y_0{}^2}\right)^2-\frac{1}{16}\left(1-\frac{b}{Y_0{}^2}\right)^3-\frac{15}{128}\left(1-\frac{b}{Y_0{}^2}\right)^4\right\}$$

Simple SQRT() Routine

- **Initial Guess**

$$
\begin{aligned}
r &= s^{\frac{1}{2}} \\
&\approx f(s) = c_0 + c_1 s + c_2 s^2 + c_3 s^3 \\
&= c_0 + s\times\left(c_1 + s\times(c_2 + s\times c_3)\right)
\end{aligned}
$$

where $0.1 < r^2 < 1.0$

$c_0 = 0.188030699;\ c_1 = 1.48359853$

$c_2 = -1.0979059;\ c_3 = 0.430357353$

- **Newton-Raphson Refinement**

$$\delta s \leftarrow s - f(s)^2$$

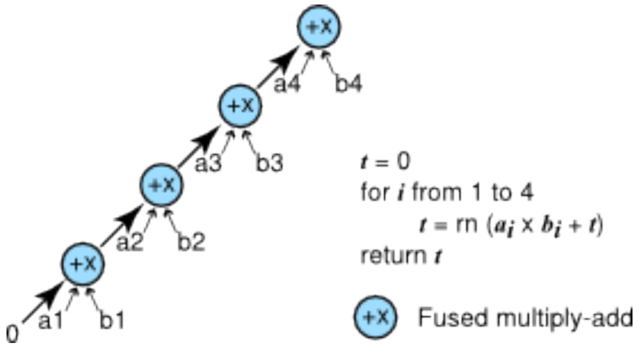
$$r \leftarrow f(s) + \delta s / 2 f(s)$$

M.P. Allen & D.J. Tildesley, *Computer Simulation of Liquids* (Oxford Univ. Press, Oxford, 1987) p.143

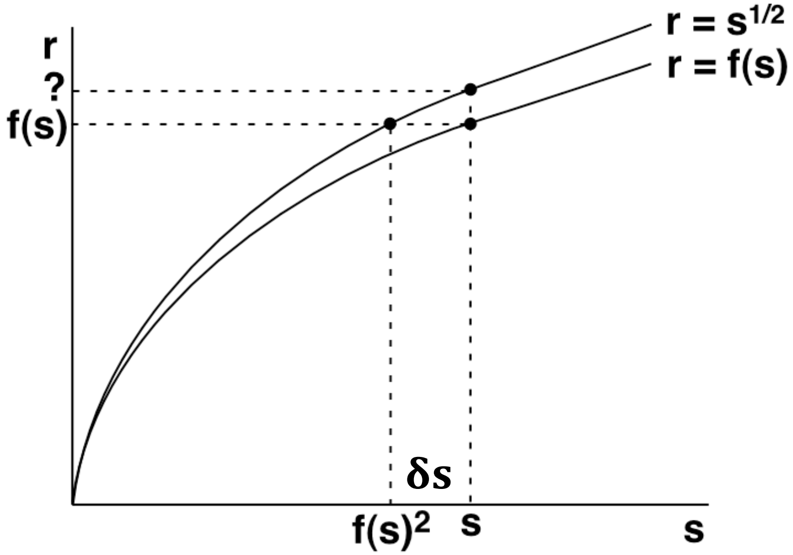
Fused multiply-add (FMA) unit

$$a \leftarrow a + b\times c$$

with 1-cycle throughput

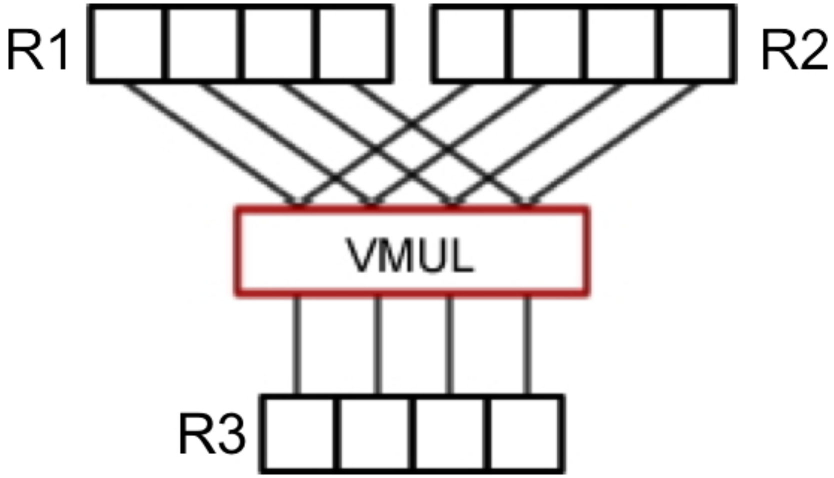


```
t = 0
for i from 1 to 4
        t = rn ($a_i$ × $b_i$ + t)
return t
```



SIMD/Vector Operation

- **Single-instruction multiple-data (SIMD) parallelism: An arithmetic operation is operated on multiple operand-pairs stored in vector registers, each of which can hold multiple double-precision numbers.**



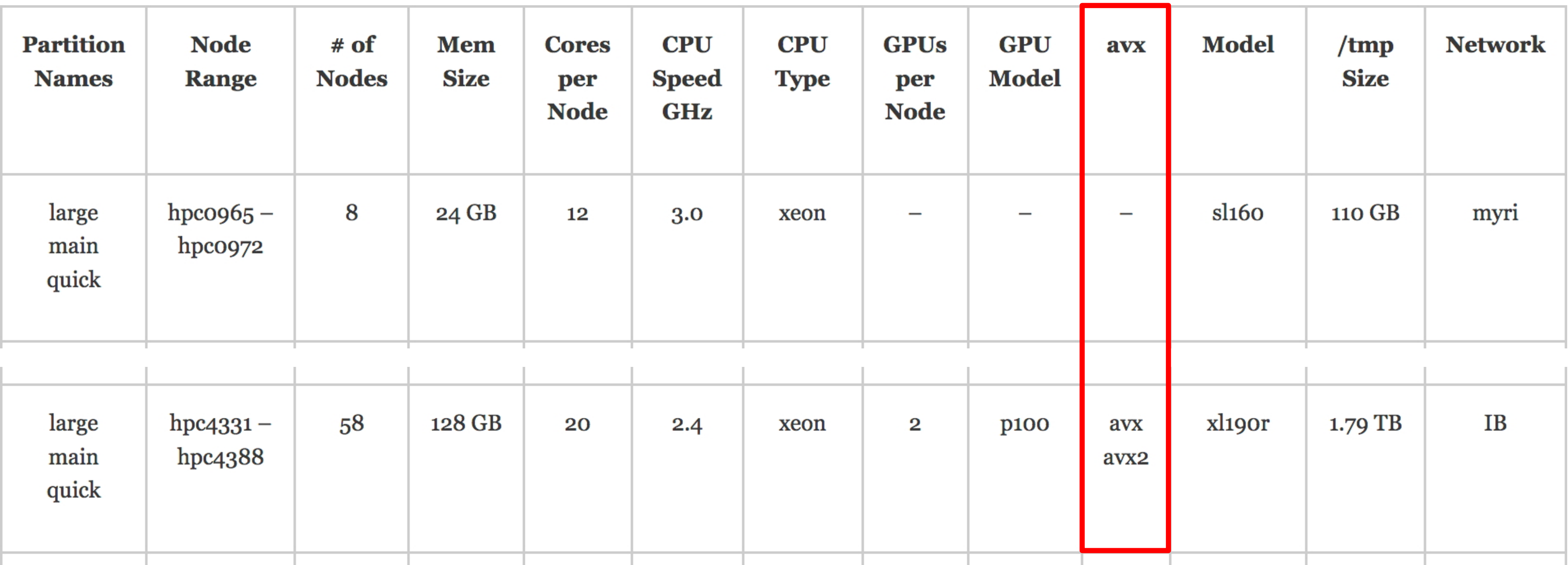
Example: Vector multiplier (VMUL) loads data from two vector registers, R1 and R2, each holding 4 double-precision numbers, concurrently performs 4 multiplications, and stores the results on vector register R3.

- **Peak performance enhancement on top of FMA.**

Vector Processing at HPC

Node information

http://hpcc.usc.edu/support/infrastructure/node-allocation



Partition Names	Node Range	# of Nodes	Mem Size	Cores per Node	CPU Speed GHz	CPU Type	GPUs per Node	GPU Model	avx	Model	/tmp Size	Network
large main quick	hpc0965 – hpc0972	8	24 GB	12	3.0	xeon	–	–	–	sl160	110 GB	myri
large main quick	hpc4331 – hpc4388	58	128 GB	20	2.4	xeon	2	p100	avx avx2	xl190r	1.79 TB	IB

Intel & AMD advanced vector extension (AVX):

- **AVX2 operates on 4 double-precision floating-point numbers**
- **AVX512 8**