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>

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 -l -q iris -t 30 -n 1
> export MODULEPATH=/soft/restricted/CNDA/modulefiles
> module load omp
> icpx -fopenmp -fopenmp-targets=spir64=-fno-exceptions led.C four1s.c -o led
>
> # GNU compiler + NVIDIA GPU test
> qsub -l -q gpu_mules -t 30 -n 1
> export MODULEPATH=$MODULEPATH:/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
FLOATING-POINT UNIT DESIGN

USING TAYLOR-SERIES EXPANSION ALGORITHMS

by

Taek-Jun Kwon

Thesis Proposal

UNIVERSITY OF SOUTHERN CALIFORNIA
ELECTRICAL ENGINEERING

September 2006
How Time Consuming Is SQRT()?

Table 1.1 Summary of prototype FPUs

<table>
<thead>
<tr>
<th>Design</th>
<th>Cycle time (ns)</th>
<th>Latency/Throughput (cycles/cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$a \pm b$</td>
</tr>
<tr>
<td>DEC 21164 Alpha AXP</td>
<td>2.0</td>
<td>4/1</td>
</tr>
<tr>
<td>Hal Sparc64</td>
<td>6.49</td>
<td>4/1</td>
</tr>
<tr>
<td>HP PA 7200</td>
<td>7.14</td>
<td>2/1</td>
</tr>
<tr>
<td>HP PA 8000</td>
<td>5.0</td>
<td>3/1</td>
</tr>
<tr>
<td>IBM RS/6000 Power 2</td>
<td>14.0</td>
<td>2/1</td>
</tr>
<tr>
<td>Intel Pentium</td>
<td>5.0</td>
<td>3/1</td>
</tr>
<tr>
<td>Intel Pentium Pro</td>
<td>7.52</td>
<td>3/1</td>
</tr>
<tr>
<td>MIPS R8000</td>
<td>13.3</td>
<td>4/1</td>
</tr>
<tr>
<td>MIPS R10000</td>
<td>3.64</td>
<td>2/1</td>
</tr>
<tr>
<td>PowerPC 604</td>
<td>5.56</td>
<td>3/1</td>
</tr>
<tr>
<td>PowerPC 620</td>
<td>7.5</td>
<td>3/1</td>
</tr>
<tr>
<td>Sun SuperSparc</td>
<td>16.7</td>
<td>3/1</td>
</tr>
<tr>
<td>Sun UltraSparc</td>
<td>4</td>
<td>3/1</td>
</tr>
</tbody>
</table>

- **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

![Image of Newton-Raphson algorithm](image1)

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**

\[
\begin{align*}
1 \quad r &= s^{2} \\
&\approx f(s) = c_0 + c_1s + c_2s^2 + c_3s^3 \\
&= c_0 + s\times(c_1 + s\times(c_2 + s\times c_3))
\end{align*}
\]

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**

\[
\begin{align*}
\delta s &\leftarrow s - f(s)^2 \\
r &\leftarrow f(s) + \delta s / 2 f(s)
\end{align*}
\]

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

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

Intel & AMD advanced vector extension (AVX):
- **AVX2** operates on 4 double-precision floating-point numbers
- **AVX512** 8