Spectral Turbulence Simulations with ClearSpeed Accelerator

Massimiliano Fatica, ClearSpeed Technology
Alan Wray, NASA Ames Research Center
Doug Enright, ClearSpeed Technology
Spectral methods are a popular tool to investigate fundamental processes in fluid dynamics.

While their applicability is limited to simple geometries, they are a powerful tool due to their accuracy and computational efficiency.

The objective of this work is to explore the use of ClearSpeed’s floating point accelerator, the CSX600, to perform spectral simulations of isotropic turbulence.
ClearSpeed’s CSX600 is a low power (10W), data-parallel co-processor with 96 cores and IEEE 754 double precision arithmetic.
Multi-Threaded Array Processing
- Programmed in high-level languages
- Hardware multi-threading for latency tolerance
- Asynchronous, overlapped I/O
- Run-time extensible instruction set
- Bi-endian (compatible with host CPU)

Array of 96 Processor Elements (PEs)
- Each is a Very Long Instruction Word (VLIW) core, not just an ALU
- Flexible data parallel processing

High performance, low power dissipation
Each PE is a VLIW core:

- Multiple execution units
  - 4-stage floating point adder
  - 4-stage floating point multiplier
  - Divide/square root unit
  - Fixed-point MAC 16x16 → 32+64
  - Integer ALU with shifter
  - Load/store
- High-bandwidth, 5-port register file (3r, 2w)
- 6 KB SRAM for data
- High bandwidth per PE DMA (PIO)
- Per PE address generators
ClearSpeed Advance™ Accelerator Board

Currently available as a PCI-X card with 2 CSX600 chips and 1 GB of memory.

- CSX600 coprocessors
- DRAM’s – 512MB per processor
- FPGA – the interface between ClearConnect & PCI-X
How to use the CSX600?

There are two ways:

1. Use the libraries provided from ClearSpeed. Initially BLAS and FFT.

2. Program the card using Cn, a C extension. ClearSpeed provides a compiler and a library for host/board interactions.
Use of libraries provided from ClearSpeed. Initially BLAS and FFT.
Cn is a C-like language for MTAP architecture:

- Modelled on the C language for familiarity
- Minimal learning curve
- Support for parallel data types: **mono** and **poly** keywords
  - **mono** is a serial (single) variable:
    » One copy exists on mono execution unit
    » Visible to all processing elements in poly execution unit
    » mono assumed unless poly specified
  - **poly** is a parallel (vector) variable:
    » One per processing element in the poly execution unit
    » Visible to a single processing element
    » Data can be shared via “swazzle” operation
    » Not visible to mono execution unit
void daxpy(double *c, double *a, double alpha, uint N) {
    uint i;
    for (i=0; i<N; i++)
        c[i] = c[i] + a[i]*alpha;
}

void daxpy(double *c, double *a, double alpha, uint N) {
    uint i;
    poly double cp, ap;
    poly int pe_num=get_penum();
    for (i=0; i<N; i+= get_numpes()) {
        memcpym2p(&cp, &c[i+pe_num], sizeof(double));
        memcpym2p(&ap, &a[i+pe_num], sizeof(double));
        cp = cp + ap*alpha;
        memcpyp2m(&c[i+pe_num], &cp, sizeof(double))
    
}
3 Tiers of Memory

1) From Host memory to Mono memory: CSAPI_Write_Mono_Memory
2) From Mono memory to Poly memory: memcpym2p or implicit broadcast
3) Compute in Poly memory
4) From Poly memory to Mono memory: memcpyp2m
5) From Mono memory to Host memory: CSAPI_Read_Mono_Memory

The host library (BLAS and FFT) performs all these steps on behalf of the user
Numerical Method

- Navier-Stokes equations in primitive variables
- Pseudo-spectral code based on Rogallo (NASA TM 81315)
- Time advancement performed with 4\textsuperscript{th} order low-storage Runge-Kutta
- De-aliasing performed by phase-shift
- Code in Fortran 90

Test case and platform selected for initial tests:

\[256^3 \text{ and } 512^3\]
Dual dual-core Xeon 2.8Ghz with 8GB of memory
ClearSpeed Advance PCI-X board

Original code with hand-coded FFT used for comparison and validation of results.
Host results

First task was converting the code to use Intel MKL FFT.

<table>
<thead>
<tr>
<th></th>
<th>Original FFTs</th>
<th>MKL FFTs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$128^3$</td>
<td>6.6 sec per time step</td>
<td>2.4 sec per time step</td>
</tr>
<tr>
<td>$256^3$</td>
<td>67.0 sec per time step</td>
<td>20.9 sec per time step</td>
</tr>
<tr>
<td>$512^3$</td>
<td>-</td>
<td>205.0 sec per time step</td>
</tr>
</tbody>
</table>

Code has been restructured to allow use of the ClearSpeed FFT library.
Most of the CPU time is spent in the computation of the non-linear terms. The velocity field is transformed from Fourier space to physical space, where the non-linear terms are computed and then transformed back to Fourier space.

To transform the data between spaces, the original code was performing a sequence of three 1D FFTs, one for each spatial direction.

To reduce the impact of the PCI-X overhead transfer time, this process has been modified to single 1D FFTs followed by 2D FFTs:

- 1D inverse FFT in y direction on host
- 2D inverse FFT in x,z directions on card
- Non-linear terms on host
- 2D FFT in x,z directions on card
- 1D FFT in y direction on host

```
Host memory       Card memory

<table>
<thead>
<tr>
<th>u,v,w</th>
<th>3</th>
<th>iFFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>NL</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

18 2D N² transfers
```
## Initial timing and power consumption

<table>
<thead>
<tr>
<th></th>
<th>Original FFTs</th>
<th>MKL FFTs</th>
<th>with Advance board</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sec per time step</td>
<td>sec per time step</td>
<td>sec per time step</td>
</tr>
<tr>
<td>$128^3$:</td>
<td>6.6s</td>
<td>2.4s</td>
<td>-</td>
</tr>
<tr>
<td>$256^3$:</td>
<td>67.6s</td>
<td>20.9s</td>
<td>24.5s</td>
</tr>
<tr>
<td>$512^3$:</td>
<td>-</td>
<td>205.0s</td>
<td>200.5s</td>
</tr>
</tbody>
</table>

Power consumption during the accelerated portion of the computation:

<table>
<thead>
<tr>
<th></th>
<th>Idle system</th>
<th>Host (MKL)</th>
<th>Host + Advance board</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>233W</td>
<td>310W</td>
<td>266W</td>
</tr>
</tbody>
</table>
Measurement of I/O overhead

The FFT time for a host call to CSDFT could be decomposed as:

- Transfer time from host memory to board memory
- Processing time on CSX600 (this include transfer of data to/from poly memory)
- Transfer time from board memory to host memory

It is possible to leave data on the board and operate on it.

<table>
<thead>
<tr>
<th>Host memory</th>
<th>Card memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(x)</td>
<td>FFT</td>
</tr>
<tr>
<td>u(k)</td>
<td>iFF</td>
</tr>
<tr>
<td>u(x)</td>
<td>FFT</td>
</tr>
<tr>
<td></td>
<td>iFF</td>
</tr>
</tbody>
</table>

4 2D N² transfers

<table>
<thead>
<tr>
<th>Host memory</th>
<th>Card memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(x)</td>
<td>FFT</td>
</tr>
<tr>
<td></td>
<td>FFT</td>
</tr>
<tr>
<td>u(x)</td>
<td>iFF</td>
</tr>
</tbody>
</table>

2 2D N² transfers

<table>
<thead>
<tr>
<th>512²</th>
<th>I/O time</th>
<th>1024²</th>
<th>FFT time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.00687s</td>
<td></td>
<td>.00675s</td>
</tr>
<tr>
<td></td>
<td>.02617s</td>
<td></td>
<td>.03005s</td>
</tr>
</tbody>
</table>
Preliminary PCIe bandwidth results

![Graph showing PCIe bandwidth results](image)
Conclusions

• Even with unnecessary traffic on the PCI-X bus, offloading the 2D transforms to the board results in performance comparable to the host CPU, while using less power.

• $256^2$ and $512^2$ problem sizes are still small for the board. $1024^2$ and $2048^2$ (for runs on clusters) will be better suited. PCIe will lower the threshold and upcoming versions of the FFT library will have improved performance.

• Better performance will be obtained by moving all the non-linear computations to the board, reducing the number of 2D arrays transferred from 18 to 9. The implementation of this strategy is under way.
Conclusions

• CSDFT API is being improved to better support operations on data resident in poly memory. This will enable an efficient computation of the non-linear terms on the board.

• In this initial work, all the 2D FFTs have been offloaded to the board. Host CPU(s) and the Advance board can be used in parallel for maximum performance.