Benchmarking and software sustainability at the exascale era

Preparing BigDFT for Aurora

Christoph Bauinger, Luigi Genovese

intel.

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), http://opensource.org/licenses/0BSD.

Motivation: Aurora

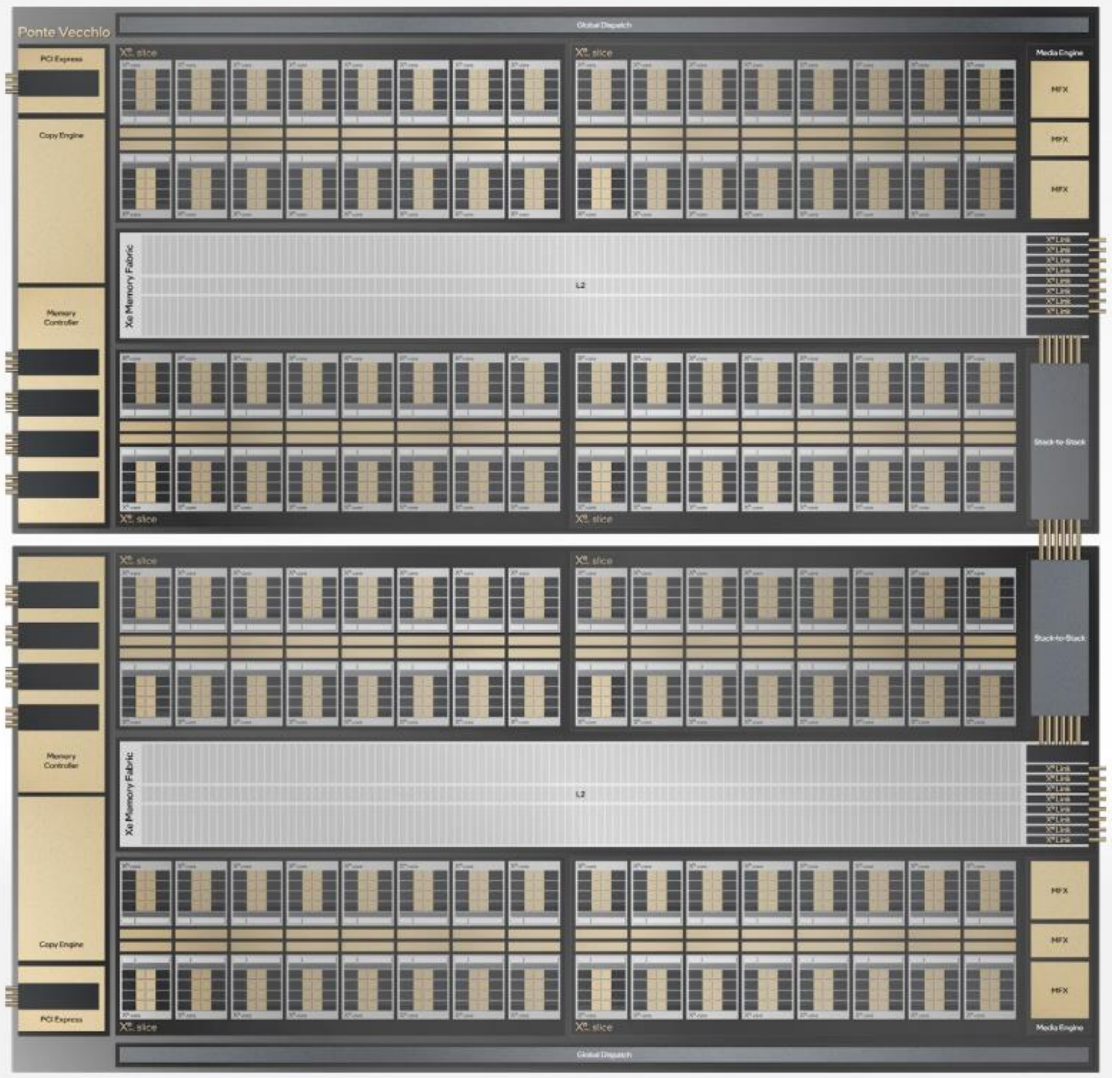
- Computer system at Argonne National Laboratory (ANL)
- >2 Exaflops FP64 theoretical peak performance
- 21,248 Intel Xeon CPU
- 63,744 Intel Data Center GPU

https://www.anl.gov/article/us-department-of-energys-incite-program-seeks-proposals-for-2024-to-advance-science-and-engineering, public domain

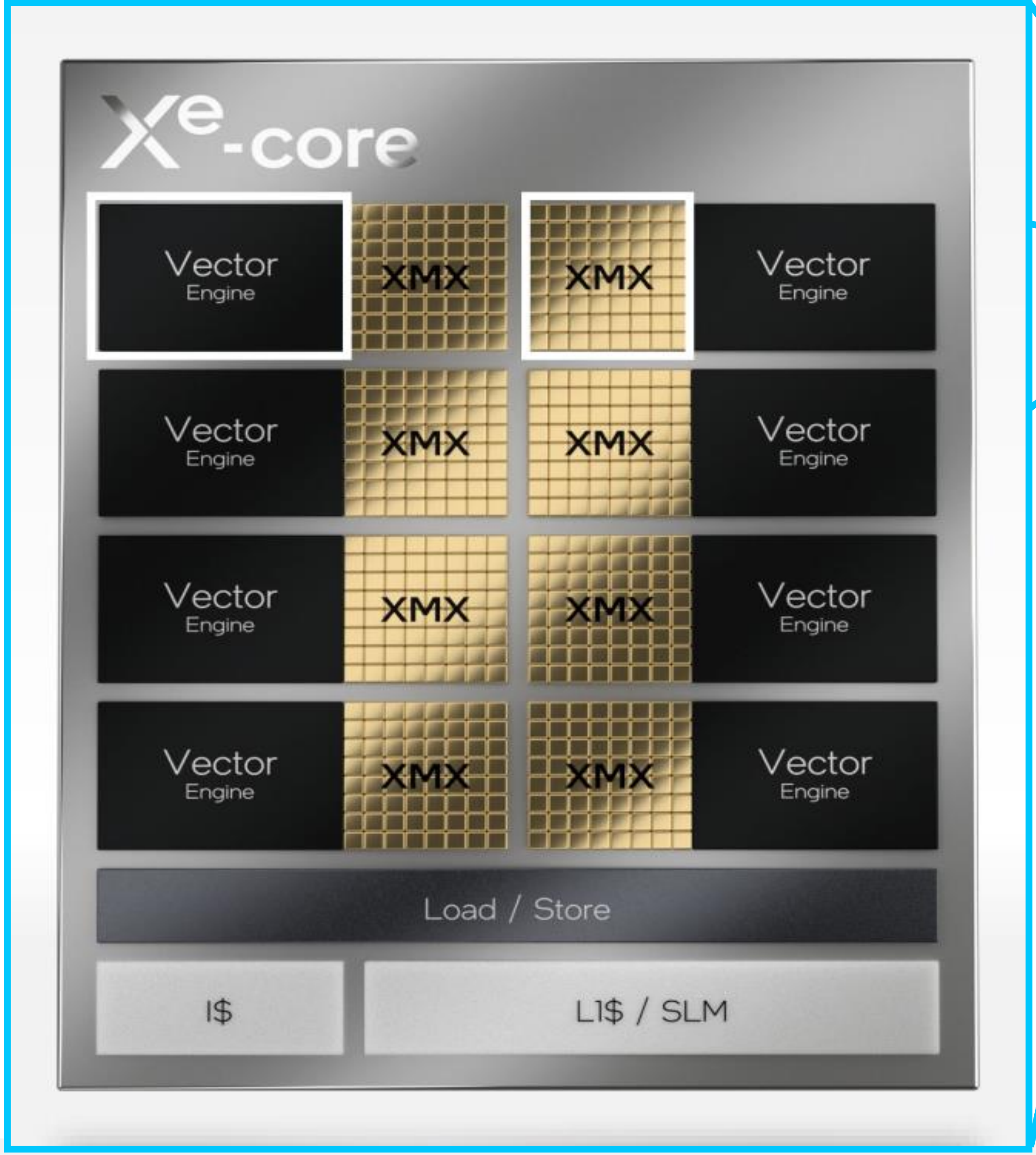
https://www.alcf.anl.gov/aurora

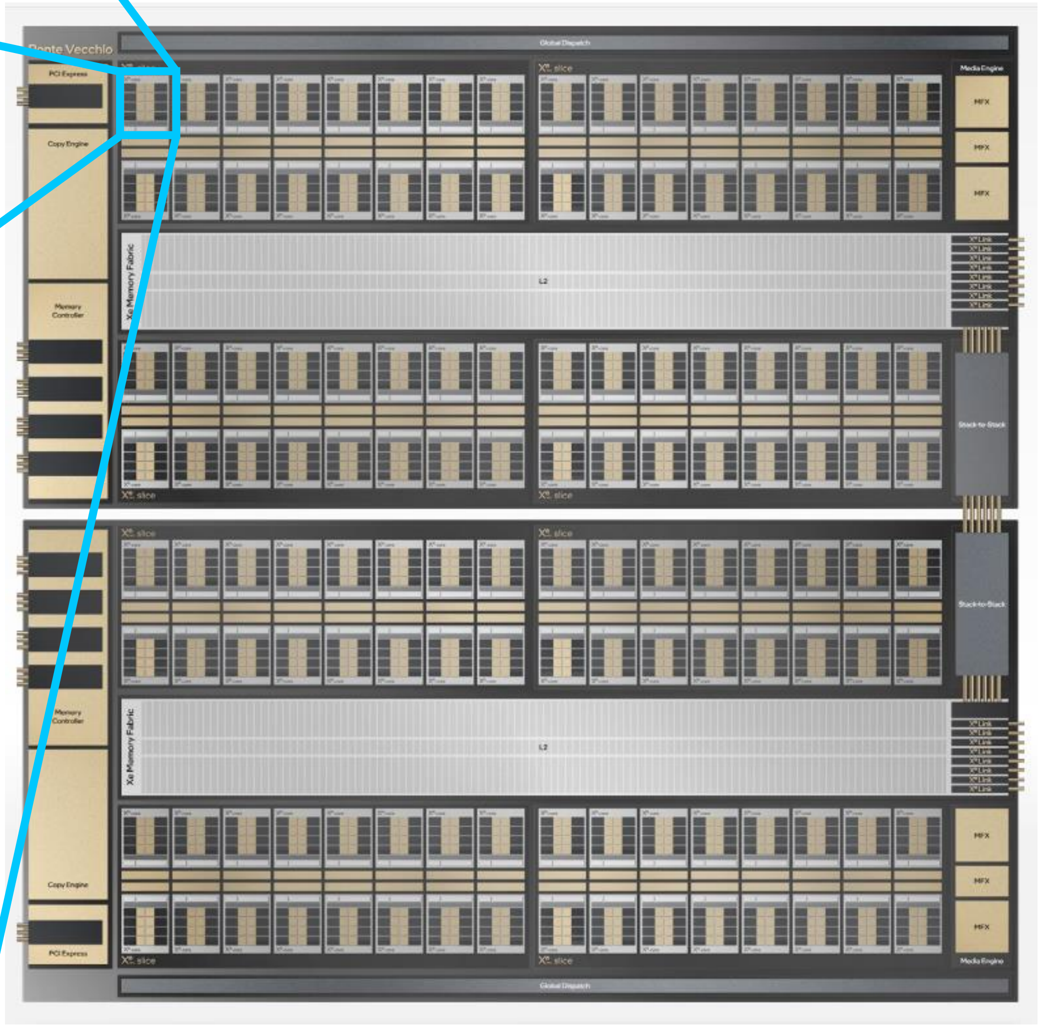
Intel Data Center GPU (PVC)

- 128GB HBM memory
- ~52 TFLOPS processing power (FP32 and FP64)
- TDP: 600W
- 2 Stacks



https://www.intel.com/content/www/us/en/docs/oneapi/optimization-guide-gpu/2023-2/intel-xe-gpu-architecture.html#XE-HPC-2-STACK-DATA-CENTER-GPU-MAX





https://www.intel.com/content/www/us/en/docs/oneapi/optimization-guide-gpu/2023-2/intel-xe-gpu-architecture.html#XE-HPC-2-STACK-DATA-CENTER-GPU-MAX

SYCL

- Standard by Khronos Group, initially released 2014
- Enables cross-platform single-source C++ code for heterogeneous and offload processors
- Implemented in Intel oneAPI DPC++
- Supported by several libraries (oneMKL, oneDNN, Kokkos, AMReX, etc.)

```cpp
#include <sycl/sycl.hpp>

int main(int argc, char ** argv)
{
    sycl::queue Q(sycl::gpu_selector_v);
    double *val_in =  sycl::malloc_shared<double>(32, Q);
    double *val_out = sycl::malloc_shared<double>(32, Q);

    Q.fill(val_in, 1.1, 32);

    Q.parallel_for(sycl::nd_range<1>(32, 32), [=](auto it)  {

        const int i = it.get_global_linear_id();

        val_out[i] = std::sqrt(val_in[i]);

    }).wait();

    sycl::free(val_in, Q);
    sycl::free(val_out, Q);

    return 0;
}
```

A simple SYCL example

BigDFT

- Code which implements Kohn-Sham Density Functional Theory (KS-DFT)
- Fortran code with C++, OpenCL, and CUDA
- “PBE0” functional GPU enabled (CUDA, OpenCL) since 2009
- CUDA implementation shown to scale to several thousand NVIDIA nodes

Laura E Ratcliff, A Degomme, José A Flores-Livas, Stefan Goedecker, and Luigi Genovese. 2018. Affordable and accurate large-scale hybrid-functional calculations on GPU-accelerated supercomputers. Journal of Physics: Condensed Matter 30, 9 (feb 2018), 095901. https://doi.org/10.1088/1361-648X/aaa8c9

Laura E Ratcliff, Luigi Genovese, Hyowon Park, Peter B Littlewood, and Alejandro Lopez-Bezanilla. 2021. “Exploring metastable states in UO2 using hybrid functionals and dynamical mean field theory”. Journal of Physics: Condensed Matter 34, 9 (dec 2021), 094003. https://doi.org/10.1088/1361-648X/ac3cf1

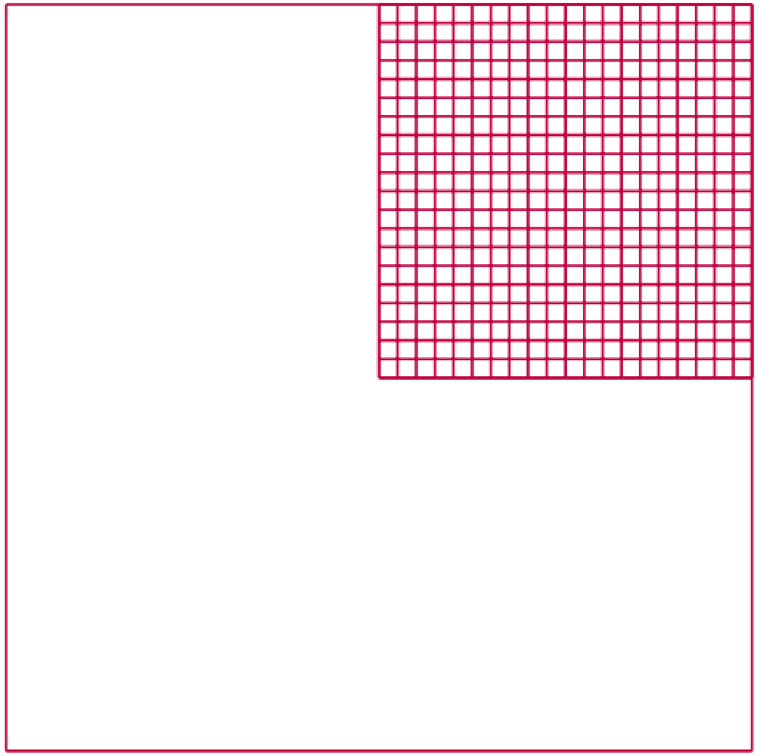
BigDFT – Fock operator evaluation

- Need to evaluate the Fock operator $$\left[\hat{D}_X\psi_i\right](\mathbf{r}) = \sum_j \underbrace{\int \mathrm{d}\mathbf{r}' \frac{\psi_j^*(\mathbf{r}')\psi_i(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}}_{V_{i,j}(\mathbf{r})}\psi_j(\mathbf{r})$$

1. Compute $\rho_{i,j}(\mathbf{r}) = \psi_j^*(\mathbf{r})\,\psi_i(\mathbf{r})$
2. Solve Poisson equation $\nabla^2 V_{i,j}(\mathbf{r}) = \rho_{i,j}(\mathbf{r})$
3. Multiply $V_{i,j}(\mathbf{r})$ with $\psi_j(\mathbf{r})$ to get Fock operator

BigDFT – Poisson Solver

- Uses convolution with Green function in Frequency Domain
- Performs convolution **as batched 1D FFTs**
- Can handle different boundary conditions
- Requires 0-padding in case of free boundary conditions



2D example of zero-padding

Nazim Dugan, Luigi Genovese, and Stefan Goedecker. 2013. A customized 3D GPU Poisson solver for free boundary conditions. Computer Physics Communications 184, 8 (2013), 1815–1820. https://doi.org/10.1016/j.cpc.2013.02.024

BigDFT – Poisson Solver

- Uses convolution with Green function in Frequency Domain
- Performs convolution **as batched 1D FFTs**
- Can handle different boundary conditions
- Requires 0-padding in case of free boundary conditions

```
if (<Boundary condition in x is free>)
    <zero-pad in x direction>
1DFFT_X(Ny*Nz, Sx) //real-to-complex FFT
if (<Boundary condition in y is free>)
    <zero-pad in y direction>
<transpose data>
1DFFT_Y((Sx/2+1)*Nz, Sy)
if (<Boundary condition in z is free>)
    <zero-pad in z direction>
<transpose data>
1DFFT_Z((Sx/2+1)*Sy, Sz)

<Convolution kernel multiplication>

inverse_1DFFT_Z((Sx/2+1)*Sy, Sz)
if (<Boundary condition in z is free>)
    <remove padding in z direction>
<inverse transpose>
inverse_1DFFT_Y((Sx/2+1)*Nz, Sy)
if (<Boundary condition in y is free>)
    <remove padding in y direction>
<inverse transpose>
inverse_1DFFT_X(Ny*Nz, Sx) //complex-to-real
if (<Boundary condition in z is free>)
    <remove padding in x direction>
```

Pseudo code of Poisson Solver in BigDFT

Nazim Dugan, Luigi Genovese, and Stefan Goedecker. 2013. A customized 3D GPU Poisson solver for free boundary conditions. Computer Physics Communications 184, 8 (2013), 1815–1820. https://doi.org/10.1016/j.cpc.2013.02.024

Porting BigDFT to SYCL – compatibility tool

- Existing accelerated code is CUDA and uses CuFFT
- We used Intel's DPC++ compatibility tool to automatically translate CUDA to SYCL
- We introduced an interface layer which chooses at runtime the CUDA or SYCL code
- oneMKL instead of CuFFT

```
//CUDA
__global__ void post_computation_kernel(int nx, int ny, int nz,
    double *rho, double *data1, int shift1, double *data2,
    int shift2, double hfac) {
    int tj = threadIdx.x;
    int td = blockDim.x;
    int blockData = (nx*ny*nz) / (gridDim.x*gridDim.y);
    int jj = (blockIdx.y*gridDim.x + blockIdx.x)*blockData;

    for (int k=0; k<blockData/td; k++) {
        int idx = jj + tj + k*td;
        data1[idx+shift1] = data1[idx+shift1] +
            hfac*rho[idx]*data2[idx+shift2];
    }
}

//SYCL
void post_computation_kernel(int nx, int ny, int nz,
    double *rho, double *data1, int shift1, double *data2,
    int shift2, double hfac, const sycl::nd_item<3> &item) {
    int tj = item.get_local_id(2);
    int td = item.get_local_range(2);
    int blockData = (nx*ny*nz) /
        (item.get_group_range(2)*item.get_group_range(1));
    int jj = (item.get_group(1)*item.get_group_range(2) +
        item.get_group(2))*blockData;

    for (int k=0; k<blockData/td; k++) {
        int idx = jj + tj + k*td;
        data1[idx+shift1] = data1[idx+shift1] +
            hfac*rho[idx]*data2[idx+shift2];
    }
}
```

CUDA kernel and automatically translated SYCL kernel.

https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compatibility-tool.html

Porting BigDFT to SYCL – BBFFT

- Goal: Further increasing SYCL performance on CPU
- Transposing and padding data is memory-bandwidth intensive and problematic on CPU
- Introduced double-batched FFT library

```
if (<Boundary condition in x is free>)
    <zero-pad in x direction>
1DFFT_X(Ny*Nz, Sx) //real-to-complex FFT
if (<Boundary condition in y is free>)
    <zero-pad in y direction>
<transpose data>
1DFFT_Y((Sx/2+1)*Nz, Sy)
if (<Boundary condition in z is free>)
    <zero-pad in z direction>
<transpose data>
1DFFT_Z((Sx/2+1)*Sy, Sz)

<Convolution kernel multiplication>

inverse_1DFFT_Z((Sx/2+1)*Sy, Sz)
if (<Boundary condition in z is free>)
    <remove padding in z direction>
<inverse transpose>
inverse_1DFFT_Y((Sx/2+1)*Nz, Sy)
if (<Boundary condition in y is free>)
    <remove padding in y direction>
<inverse transpose>
inverse_1DFFT_X(Ny*Nz, Sx) //complex-to-real
if (<Boundary condition in z is free>)
    <remove padding in x direction>
```

Remove additional memory accesses, where applicable

https://github.com/intel/double-batched-fft-library

Porting BigDFT to SYCL – BBFFT

- Bbfft allows zero-padding on-the-fly with callbacks
- Avoids transposing data due to double-batching
- In addition, utilize symmetries to half memory requirements for kernel
- Halved memory requirements
- More than halved computing time on 4^{th} generation Xeon CPU

```
if (<Boundary condition in x is free>)
    <set callback to add zeros>
1DFFT_X(Ny*Nz, Sx) //real-to-complex FFT
if (<Boundary condition in y is free>)
    <set callback to add zeros>
1DFFT_Y((Sx/2+1)*Nz, Sy)
if (<Boundary condition in z is free>)
    <set callback to add zeros>
1DFFT_Z((Sx/2+1)*Sy, Sz)

<Convolution kernel multiplication>

inverse_1DFFT_Z((Sx/2+1)*Sy, Sz)
inverse_1DFFT_Y((Sx/2+1)*Nz, Sy)
inverse_1DFFT_X(Ny*Nz, Sx) //complex-to-real
```

Pseudo code of Poisson Solver with bbfft in BigDFT

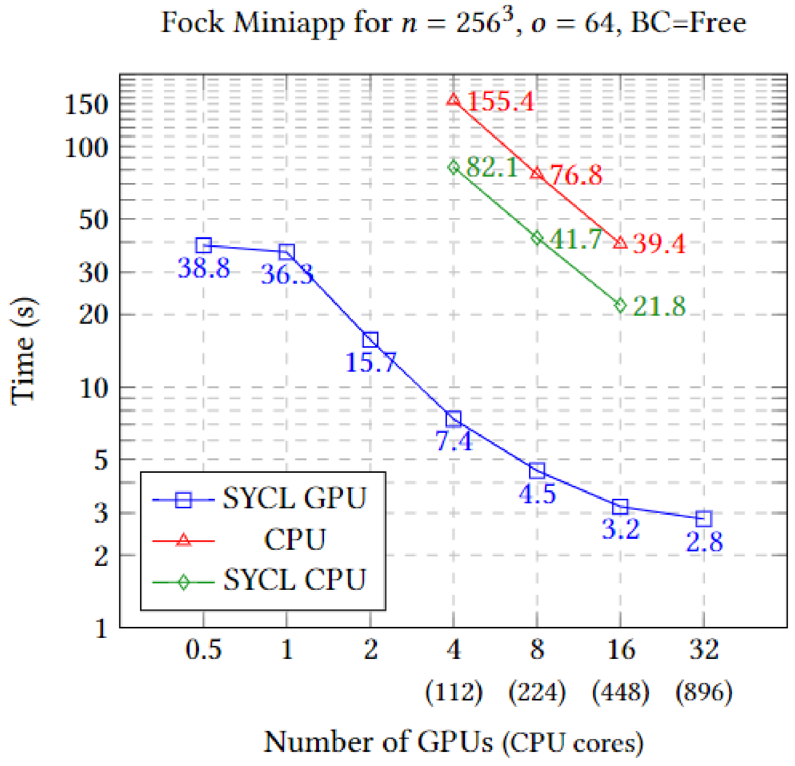
Benchmarking results – Test systems

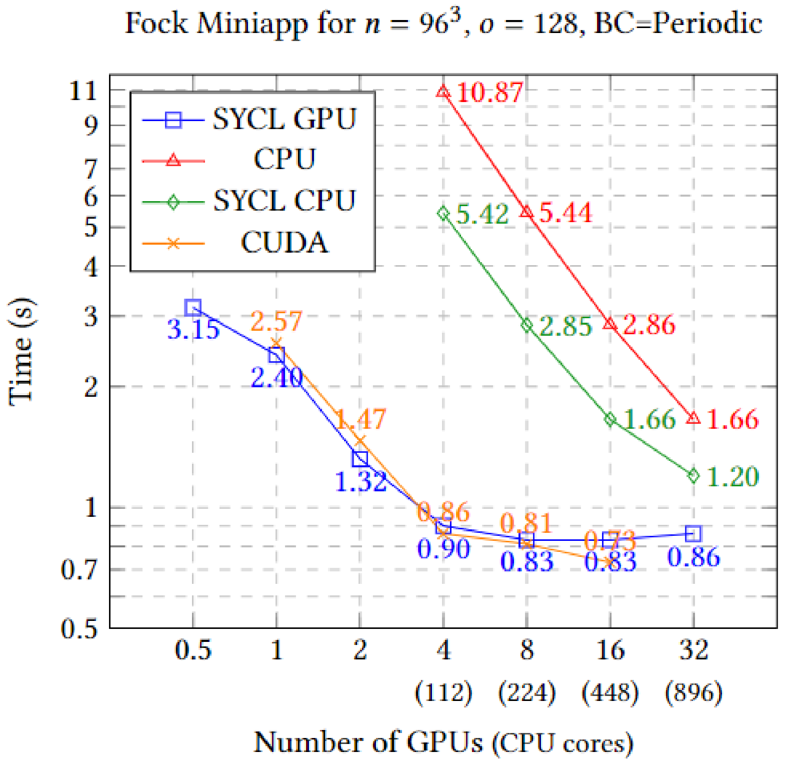
- PVC Test system: Florence
  - Lenovo SD650-I V3 servers with 2 Intel Xeon Platinum 8480+ Processors and 4 Intel Data Center GPU Max Series, 512 GB RAM
  - Each PVC is power-capped to 450 W
  - GPUs connected with XeLink
- NVIDIA Test system:
  - 2 Intel Xeon Platinum 8360Y Processors and 2 NVIDIA A100 40GB GPUs, 256 GB RAM
  - No NVLink
- Host code compiled with ifx, icx, icpx, intel oneAPI 2023.1
- Intel MPI 2021.9 in all cases

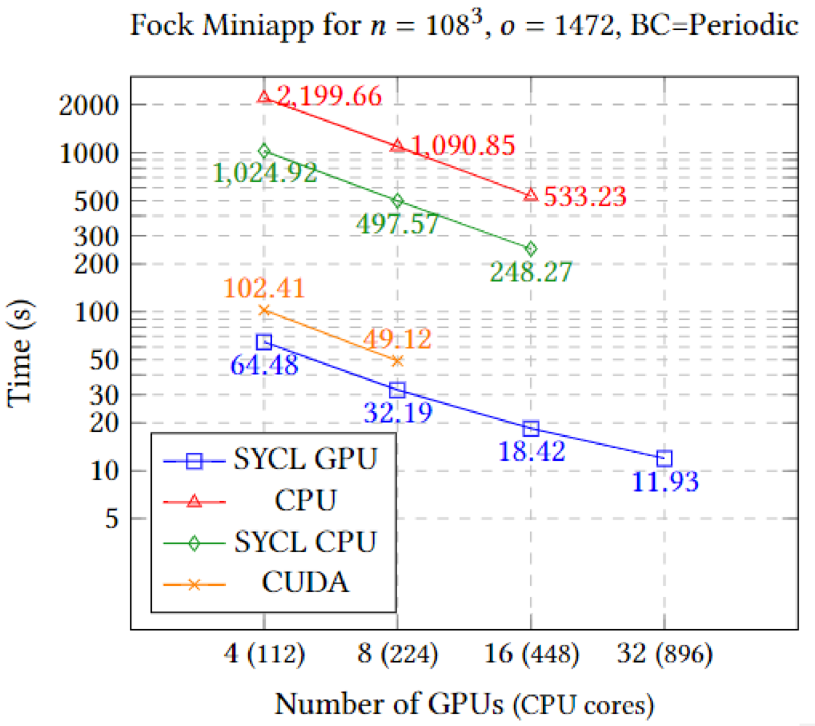
https://ark.intel.com/content/www/us/en/ark/products/231746/intel-xeon-platinum-8480-processor-105m-cache-2-00-ghz.html
https://www.intel.com/content/www/us/en/developer/articles/technical/intel-data-center-gpu-max-series-overview.html?wapkw=tuscany

Benchmarking results – Fock miniapp

- Fock miniapp is a small test code to evaluate performance and correctness of the Fock operator evaluation

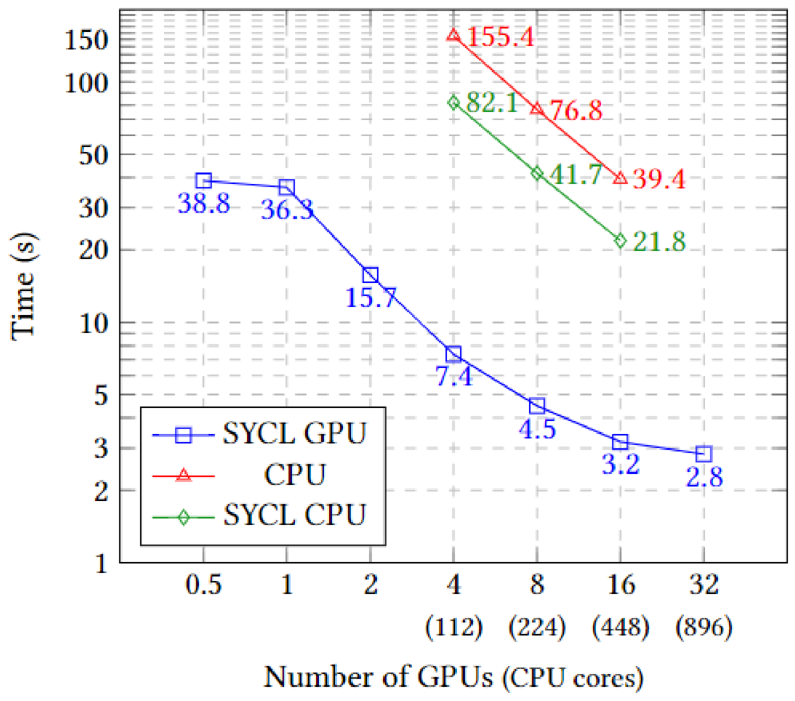




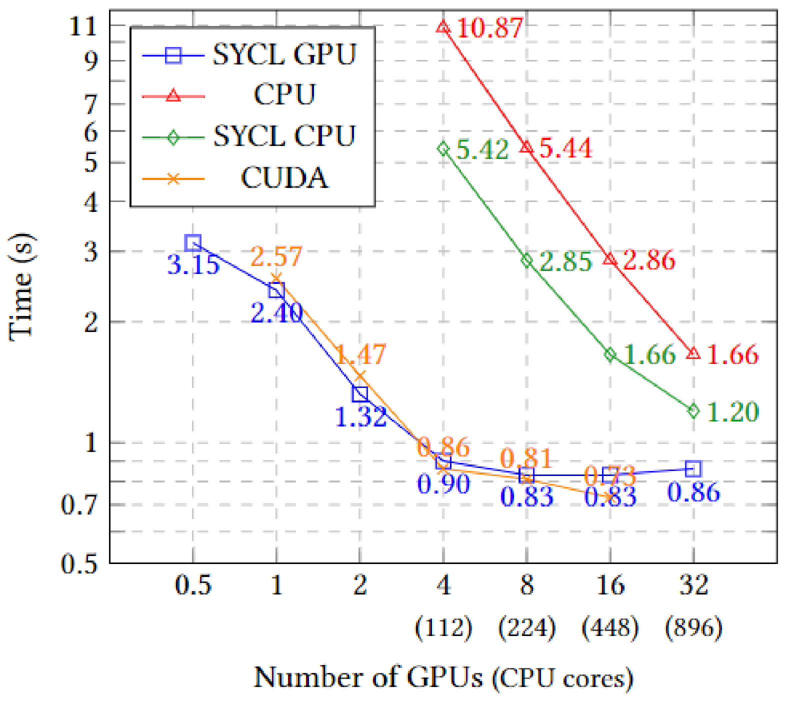


Benchmarking results – Fock miniapp

Fock Miniapp for $n = 256^3$, $o = 64$, BC=Free

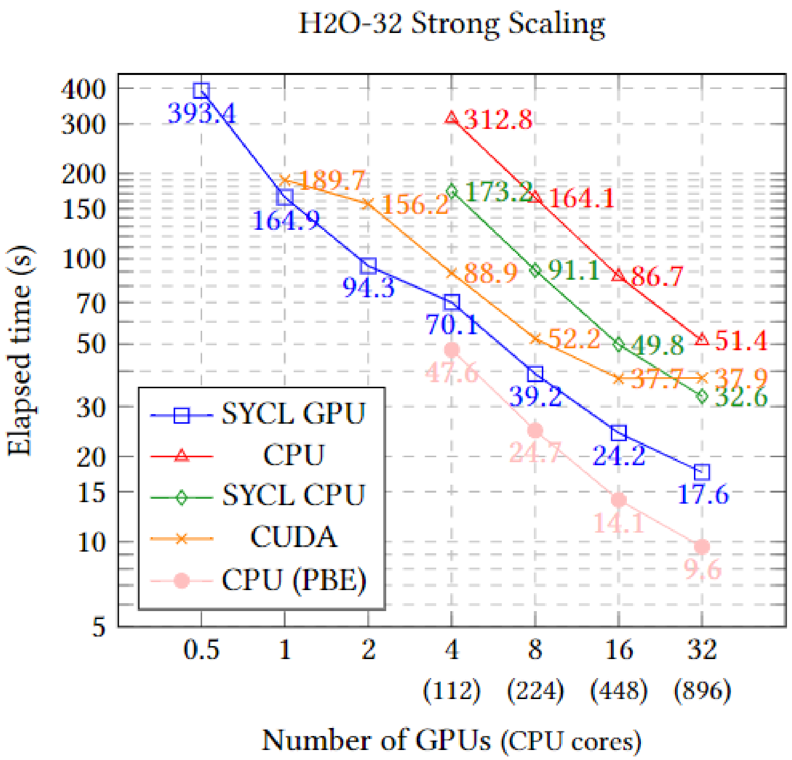


Fock Miniapp for $n = 96^3$, $o = 128$, BC=Periodic

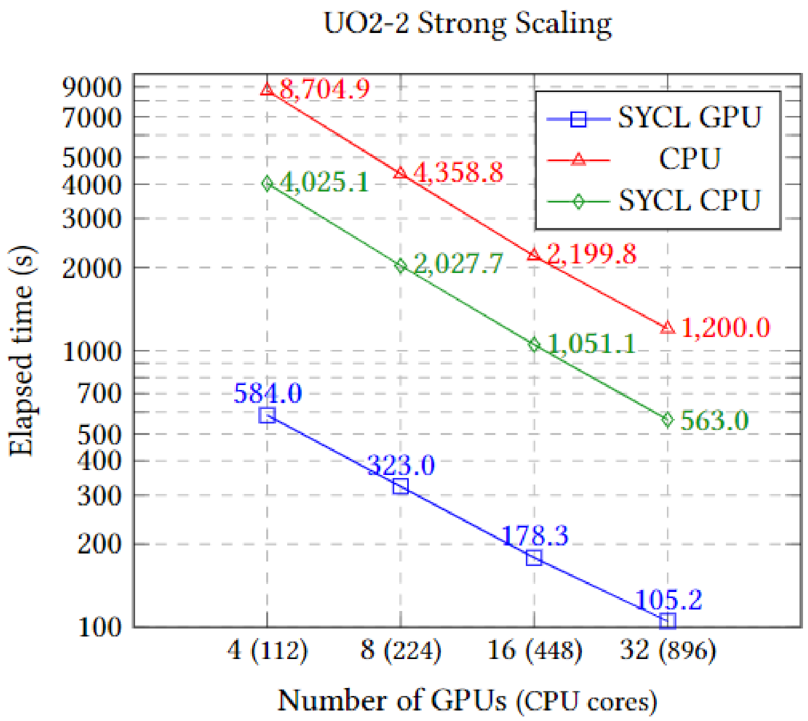


Case	Read (GB/s)		Write (GB/s)		Read+Write	
	L3	HBM	L3	HBM	L3	HBM
$n = 256^3$, $o = 64$	0.02	468	336	315	336	783
$n = 96^3$, $o = 128$	1172	66	1744	48	3516	114

Benchmarking results – Full App



32 H2O molecules. Performance delta between NVIDIA and Intel GPUs purely on the CPU side due to different processors



UO2 molecules with 1432 orbitals and a grid size of n=108x108x108.
Elapsed time is for 3 iterations.

Performance Portability

- Goal: Run SYCL code on all backends with great performance
  - Intel CPU
  - Intel GPU
  - AMD CPU
  - AMD GPU
  - NVIDIA GPU
- Problem: Lacking support from bbfft and oneMKL

Summary and Future Work

- Shown that CUDA to SYCL port is easy even in big projects
- Shown that SYCL+PVC performs and scales well, indicating readiness for Aurora
- Shown that SYCL+CPU is a viable alternative to existing Fortran+OMP implementation

- Continue working on AMD and NVIDIA support with our SYCL implementation
- Run large simulations on Aurora

intel®