Massively scalable stencil algorithm

Mathias Jacquelin
Cerebras Systems Inc.
Sunnyvale, California, USA
mathias.jacquelin@cerebras.net

Mauricio Araya-Polo and Jie Meng
TotalEnergies EP Research & Technology US, LLC.
Houston, Texas, USA
mauricio.araya@totalenergies.com

Abstract—Stencil computations lie at the heart of many scientific and industrial applications. Unfortunately, stencil algorithms perform poorly on machines with cache based memory hierarchy, due to low reuse of memory accesses. This work shows that for stencil computation a novel algorithm that leverages a localized communication strategy effectively exploits the Cerebras WSE-2, which has no cache hierarchy. This study focuses on a 25-point stencil finite-difference method for the 3D wave equation, a kernel frequently used in earth modeling as numerical simulation. In essence, the algorithm trades memory accesses for data communication and takes advantage of the fast communication fabric provided by the architecture. The algorithm—historically memory bound—becomes compute bound. This allows the implementation to achieve near perfect weak scaling, reaching up to 503 TFLOPs on WSE-2, a figure that only full clusters can eventually yield.

Index Terms—Stencil computation, high performance computing, energy, wafer-scale, distributed memory, multi-processor architecture and micro-architecture

I. INTRODUCTION

Stencil computations are central to many scientific problems and industrial applications, from weather forecast ([32]) to earthquake modeling ([19]). The memory access pattern of this kind of algorithm, in which all values in memory are accessed but used in only very few arithmetic operations, is particularly unfriendly to hierarchical memory systems of traditional architectures. Optimizing these memory operations is the main focus of performance improvement research on the topic.

Subsurface characterization is another area where stencils are widely used. The objective is to identify major structures in the subsurface that can either hold hydrocarbon or be used for CO₂ sequestration. One step towards that end is called seismic modeling, where artificial perturbations of the subsurface are modeled solving the wave equation for given initial and boundary conditions. Solving seismic modeling efficiently is crucial for subsurface characterization, since many perturbation sources need to be modeled as the subsurface model iteratively improves. The numerical simulations required by seismic algorithms for field data are extremely demanding, falling naturally in the HPC category and requiring practical evaluation of technologies and advanced hardware architectures to speed up computations.

Advances in hardware architectures have motivated algorithmic changes and optimizations to stencil applications for at least 20 years ([23]). Unfortunately, the hierarchical memory systems of most current architectures is not well-suited to stencil applications, therefore limiting performance. This applies to multi-core machines, clusters of multi-cores, and accelerator-based platforms such as GPGPUs, FPGAs, etc. ([2], [5]). Alternatively, non-hierarchical architectures were explored in this context, such as the IBM Cell BE ([3]), yielding high computational efficiency but with limited impact.

A key element for large scale simulations is the potential of deploying substantial number of processing units connected by an efficient fabric. The Cell BE lacked the former and it had limited connectivity. Another example of non-hierarchical memory system is the Connection Machine ([12]), which excelled on scaling but at the cost of a very complex connectivity. In this work, a novel stencil algorithm based on localized communications that does not depend on memory hierarchy optimizations is introduced. This algorithm can take advantage of architectures such as the WSE from Cerebras ([4]) and potentially Anton 3-like systems ([28]). These are examples of architectures addressing both limitations described above.

Another angle to be considered is the availability of hardware-based solutions in the market. Literature review yields no generally available hardware architecture addressing the specific bottlenecks of stencil applications. Only a few custom designs examples are available ([10], [14]).

In this work, an implementation of such seismic modeling method on a novel architecture is presented. The proposed mapping requires a complete redesign of the basic stencil algorithm. The contribution of this work is multi-fold:

\section*{TABLE I: Equivalences between traditional architectures and the WSE}

| Traditional architecture | WSE |
|--------------------------|-----|
| L1                       | Memory |
| L2 & L3                  |ashes |
| DRAM                     |∅     |
| Off-node interconnect    |Fabric & routers |
The paper is organized as follows: Section II reviews relevant contributions in the literature. Section III describes the target application. Section IV provides details of how the target application was redesigned to efficiently use the novel processor architecture. Sections V and VI discuss experimental results and profiling data. Section VII provides discussions and conclusions.

II. RELATED WORK

A. Stencil Computation

Not all stencil computations are the same, and the structure and order of the stencil set the limits of the attainable performance. The higher the order (neighbors to be accounted for) and the closer to a pure star shape is, the harder to compute the stencil is. Traditional hierarchical memory subsystems will be overwhelmed by the memory access pattern which displays very little data reuse. Considerable amount of research effort has been devoted to optimizing stencil computations, and to finding ways around these issues. Spurred by emerging hardware technologies, studies on how stencil algorithms can be tailored to fully exploit unique hardware characteristics have covered many aspects, from DSLs, performance modeling, to pure algorithmic optimizations, targeting low-level architectural features in some cases.

Domain-specific languages (DSLs), domain-specific parallel programming models, and compiler optimizations for stencils have been proposed (e.g., [9], [11], [16], [22]). Performance models have been developed for this computing pattern (see [6], [30]), and the kernel has been ported to a variety of platforms ( [2], [3], [5], [35]) including specific techniques to benefit from unique hardware features.

Stencil computations have also been the subject of multiple algorithmic optimizations. Spatial and temporal blocking has been proposed [8], [13], [34]. A further example is the semi-stencil algorithm proposed by De la Cruz et al. [7], which offers an improved memory access pattern and a higher level of data reuse. Promising results are also achieved using a higher dimension cache optimization, as introduced by Nguyen et al. [20], accommodating both thread-level and data-level parallelism. Most recently, Sai et al. [26] studied high-order stencils with a manually crafted collection of implementations of a 25-point seismic modeling stencil in CUDA and HIP for the latest GPGPU hardware. Along this line of hardware-oriented stencil optimizations, Matsumura et al. [17] proposed a framework (AN5D) for GPU stencil optimization, obtaining remarkable results.

Wafer-scale computations have first been explored in Rocki et al. [24], in which the authors explore a BiCGStab implementation to solve a linear system arising from a 7-point finite difference stencil on the first generation of Cerebras Wafer-Scale Engine. Albeit computing a much simpler stencil and having a higher arithmetic intensity, this study paved the way to the work presented in this study. A notable difference between the current work and this study is that floating point operations were performed in mixed precision: stencil and AXPY operations being computed using 16 bit floating point operations and global reductions using 32 bit arithmetic. In the present study, only 32 bit floating point arithmetic is used, and neither AXPY operation nor global reductions are involved. This makes the performance achieved by these two applications not directly comparable.

III. FINITE DIFFERENCE FOR SEISMIC MODELING

Minimod is a proxy application that simulates the propagation of waves through the Earth models, by solving a Finite Difference (FD) which is discretized form of the wave equation. It is designed and developed by TotalEnergies EP Research & Technologies [18]. Minimod is self-contained and designed to be portable across multiple compilers. The application suite provides both non-optimized and optimized versions of computational kernels for targeted platforms. The main purpose is benchmarking of emerging new hardware and programming technologies. Non-optimized versions are provided to allow analysis of pure compiler-based optimizations.

In this work, one of the kernels contained in Minimod is used as target for redesign: the acoustic isotropic kernel in a constant-density domain [21]. For this kernel, the wave equation PDE has the following form:

\[
\frac{1}{V^2} \frac{\partial^2 u}{\partial t^2} - \nabla^2 u = f, \tag{1}
\]

where \( u = u(x, y, z) \) is the wavefield, \( V \) is the Earth model (with velocity as the main property), and \( f \) is the source perturbation. The equation is discretized in time using a 2\textsuperscript{nd} order centered stencil, resulting in the semi-discretized equation:

\[
u^{n+1} - Qu^n + u^{n-1} = (\Delta t^2) V^2 f^n, \tag{2}
\]

with \( Q = 2 + \Delta t^2 V^2 \nabla^2 \).

Finally, the equation is discretized in space using a 25-point stencil in 3D (8\textsuperscript{th} order in space), with four points in each direction as well as the centre point:

\[
\nabla^2 u(x, y, z) \approx \sum_{m=0}^{4} c_{xm} [u(i + m, j, k) + u(i - m, j, k)] + c_{ym} [u(i, j + m, k) + u(i, j - m, k)] + c_{zm} [u(i, j, k + m) + u(i, j, k - m)].
\]
where \( c_{xm}, c_{ym}, c_{zm} \) are discretization parameters, solved in step 4 in Algorithm 1. In the remainder of the document we refer to this operator as the Laplacian.

A simulation in Minimod consists of solving the wave equation at each timestep for thousands of timesteps. Pseudocode of the algorithm is shown in Algorithm 1.

```
Data: f: source
Result: u^n: wavefield at timestep n, for n ← 1 to T
1  u^0 := 0;
2  for n ← 1 to T do
3      for each point in wavefield u^n do
4          Solve Eq. 2 (left hand side) for wavefield u^n;
5      end
6      u^n = u^n + f^n (Eq. 2 right hand side);
7  end

Algorithm 1: Minimod high-level description
```

We note that a full simulation includes additional kernels, such as I/O and boundary conditions. These additional kernels are not evaluated in this study but will be added in the future. The kernel has been ported and optimized for GPGPUs, including NVIDIA A100, full report can be found in [25], this implementation is used as baseline to compare the results with the proposed implementation.

IV. FINITE-DIFFERENCES ON THE WSE

This section introduces general architectural details of the Cerebras Wafer-Scale Engine (WSE) and discusses hardware features allowing the target application to reach the highest level of performance. The mapping of the target algorithm onto the system is then discussed. The implementation is referred to as Finite Differences in the remainder of the study. Communication strategy and core computational parts involved in Finite Differences are also reviewed.

The implementation of Finite Differences on the WSE is written in Cerebras Software Language (CSL) using the Cerebras SDK [27], which allows software developers to write custom programs for Cerebras systems. CSL is a C-like language based on Zig [31], a reinterpretation of C which provides a simpler syntax and allows to declare compile-time blocks/optimizations explicitly (rather than relying on macros and the C preprocessor). CSL provides direct access to key hardware features, while allowing the use of higher-level constructs such as functions and while loops. The language allows to express computations and communications across multiple cores. Excerpts provided in the following will use the CSL syntax.

A. The WSE architecture

The WSE is an unprecedented-scale manycore processor. It is the first wafer-scale system [4], embedding all compute and memory resources within a single silicon wafer, together with a high performance communication interconnect. An overview of the architecture is given in Figure 2. In its latest version, the WSE-2 provides a total of 850,000 processing elements, each with 48 KB of dedicated SRAM memory; up to eight 16-bit floating point operations per cycle; 16 bytes of read and 8 bytes of write bandwidth to the memory per cycle; and a 2D mesh interconnection fabric that can handle 4 bytes of bandwidth per PE per cycle in steady state [15].

The WSE can be seen as an on-wafer distributed-memory machine with a 2D-mesh interconnection fabric. This on-wafer network connects processing elements or PEs. Each PE has a very fast local memory and is connected to a router. The routers link to the routers of the four neighboring PEs. There is no shared memory. The WSE contains a 7 × 12 array of identical “dies”, each holding thousands of PEs. Other chips are made by cutting the wafer into individual die. In the WSE, however, the interconnect is extended between dies. This results in a wafer-scale processor tens of times larger than the largest processors on the market at the time of its release.

The instruction set of the WSE is designed to operate on vectors or higher dimensionality objects. This is done by using data structure descriptors, which contain information regarding how a particular object should be accessed and operated on (such as address, length, stride, etc.).

As mentioned above, given the distributed memory nature of the WSE, the interconnect plays a crucial role in delivering performance. It is convenient to think of the 2D mesh interconnect in terms of cardinal directions. Each PE has 5 full-duplex links managed by its local router: East, West, North, and South links allow data to reach other routers and PEs, while the ramp link allows data to flow between the router and the PE, on which computations can take place. Each link is able to move a 32 bit packet in each direction per cycle. Each unidirectional link operates in an independent fashion, allowing concurrent flow of data in multiple directions.

Every 32 bit packet has a color (contained in additional bits of metadata). The role of colors is twofold:

1) Colors are used in the routing of communications. A color determines the packet’s progress at each router it encounters from source to destination(s). A router controls, for each color, where — to what subset of the five links — to send a packet of that color. Moreover, colors are akin to virtual channels in which ordering is guaranteed.

2) Colors can also be used to indicate the type of a message: a color can be associated to a handler triggered when a packet of that particular color arrives.

The WSE is an unconventional parallel computing machine in the sense that the entire distributed memory machine lies within the same wafer. There is no cache hierarchy nor shared memory. Equivalences between hardware features of the WSE and what they correspond to
B. Target Algorithm/application mapping

The sheer scale of the WSE calls for a novel mapping of the target algorithm onto the processor. The 3D $nx \times ny \times nz$ grid on which the stencil computation is performed is mapped onto the WSE in two different ways: the X and Y dimensions are mapped onto the fabric while the Z dimension is mapped into the local memory of a PE. This follows the approach that was explored in Rocki et al. [24], and has the benefit of expressing the highest possible level of concurrency for this particular application. Figure 3a depicts how the domain is distributed over the WSE. Each PE owns a subset of $nz$ cells of the original grid, as depicted in Figure 3b. In order to simplify the implementation, this local subset is extended by 8 extra cells: 4 cells below and 4 cells above the actual grid. This ensures that any cell in the original grid always has 4 neighbors below and 4 neighbors above. A PE stores the wavefield at two time steps (see Equation 2). In order to lower overheads, computations and communications are performed on blocks of size $b$. The block size is chosen to be the largest such that the $2 \times (nz + 8)$ cells and all buffers depending on $b$ can fit in memory.

C. Stencil computation

Computing the spatial component (referred to as the Laplacian) of the governing PDE lies at the heart of the target application, and it is traditionally the most demanding component. In addition to requiring a significant amount of floating point operations, computing the Laplacian also involves data movement, which is known to be very expensive on distributed memory platforms.

In the context of this paper, a 25-point stencil (depicted in Figure 1) is used. The stencil spans over all three dimensions of the grid. In order to compute a particular cell, data from neighboring cells is needed in all three dimensions. More precisely, a $cell_{x,y,z}$ requires data from neighboring cells:

\[
\begin{align*}
    cell_{x-4 \leq i < x, y, z}, \\
    cell_{x < i \leq x+4, y, z}, \\
    cell_{i, y-4 \leq j < y, z}, \\
    cell_{i, y < j \leq y+4, z}, \\
    cell_{i, j, z-4 \leq k < z}, \\
    cell_{i, j, z < k \leq z+4}.
\end{align*}
\]

1) Localized broadcast patterns: Dimensions X and Y from the grid are mapped onto the fabric of the WSE. To compute the stencil, a PE has therefore to communicate with 4 of its neighbors along each cardinal direction of the PE grid. A communication strategy similar to Rocki et al. [24], in which a single color is used per neighboring PE, would have resulted in an excessive color use for the stencil of interest to this application. In this work, localized broadcast patterns along every PE grid directions (Eastbound and Westbound for the X dimension, and Northbound and Southbound for the Y dimension) are used instead. Each broadcast pattern uses two dedicated colors (one for receiving data, one for sending data) and can happen concurrently with others broadcast patterns using separate links to communicate between PEs. Given the stencil size used in the application and the number of colors available on the hardware, the limited color usage per broadcast pattern is critical to the feasibility of the implementation.

In each broadcast pattern, multiple Root PEs send their local block of data of length $b$ to their respective neighboring 4 processing elements. This pattern is depicted in Figure 4b for the Eastward direction.

The router of each PE is configured to control how packets are received and transmitted. Each router determines, for each color, the incoming links from which that color can be received and the subset of the five outgoing links to which that color will be sent. The routing can be changed at run-time by special commands which can be sent just as other packets are sent. This capability lies at the heart of the communication strategy proposed here. In Figure 4a, the different router configurations used by Finite Differences are given. All Root PEs are in configuration 0. Intermediate PEs in each broadcast are in configuration 1, while Last PEs are in configuration 2. Ideally, a Root PE should broadcast its data only to other PEs. However, due to hardware constraints, a Root PE is obliged to receive its own data as well.

After sending its local data, a Root PE sends a command to its local router and the following 4 routers. In effect, this routing update shifts the communication pattern by one position: the first neighbor now becomes a Root in the next step of the broadcast pattern. After 5 steps (and 5 shifts), a PE has sent its data out and has received data from its 4 neighbors. In Figure 4b, the target PE receives data from the West during the first 4 steps, and sends its data to the East at step 5.

One of the very important aspects of this is that changing the routing on a remote router does not require any
16 neighboring PEs along the $X$ and $Y$ directions must be exchanged. This means that at each time step, a PE is involved in 4 localized broadcast patterns (one per cardinal direction). In each broadcast pattern, a PE sends its data and receives data from 4 neighbors.

Using a FMUL instruction, incoming cells from a given direction are multiplied “on the fly” with coefficients depending on their respective distance to the local cell. There are 4 FMUL operations happening concurrently (one per cardinal direction). This is depicted as step 1 in Figure 5 for the data coming from the West. Each FMUL instruction operates on $5 \times b$ incoming cells coming from a particular cardinal direction, and the coefficients (corresponding to $c_{xm}$ and $c_{ym}$, $\forall m \in \{1\ldots4\}$ in Section III). A given coefficient is applied to $b$ consecutive cells are they are coming from the same distance neighbor.

Since a PE is receiving data from itself, it is advantageous to compute the contribution from the center $cell_{x,y,z}$ during this step. This is done during the FMUL operation that processes the cells coming from the West, by multiplying the cells coming from the same PE with $c_{x0} + c_{y0} + c_{z0}$. FMULs operating on cells coming from all other directions use a coefficient of 0 for the data coming from the same PE.

Once this distributed computation phase is complete, the data of size $4 \times 5 \times b$ is reduced into a single buffer of $b$ cells (which is referred to as accumulator) using a FADD instruction (step 2 in Figure 5). The dimension of size 4 corresponds to the number of localized broadcast patterns a PE participates in, 5 corresponds to the number of PE it is receiving from, and $b$ is the number of elements coming from each PE. All contributions of neighboring cells along the $X$ and $Y$ dimensions are contained in the accumulator buffer after the reduction.
(a) WSE-2 router configurations used by Finite Differences. Configuration 0 corresponds to the configuration of the Root of a broadcast, configuration 1 is used by PEs in the middle, configuration 2 is used by the Last PE.

(b) 5 communication steps required to fetch all the data required by a target PE from the West (steps 1 through 4) and to send its data to the East (step 5). Corresponding router configurations are given in the circled numbers. At each step, a router command is sent through the broadcast pattern, changing the configurations of each set of 5 routers.

Fig. 4: Eastward localized broadcast operation used in Finite Differences to exchange cells along the X dimension.

Fig. 5: A summary of main operations: computing the stencil over the X and Y dimensions (for each cardinal direction), reducing the accumulator buffer, and subtracting the accumulator from the wavefield.

3) Stencil computation over the Z dimension: After remote contributions to the Laplacian from the X and Y axes of the grid have been accumulated, contributions from Z can be computed. Given the problem mapping over the WSE, this means that, at each time step, the computation over the Z dimension can be performed in an embarrassingly parallel fashion since this dimension resides entirely in the memory of a PE.

Each PE executes 8 FMACs instructions of length $b$, multiplying the wavefield by one of the 8 coefficients (corresponding to discretization parameters $c_{zm}$, $\forall m \in 1 \ldots 4$). The result of each FMAC is placed into the accumulator buffer (which also contains the contributions from the X and Y dimensions). Given a target block of size $b$ starting at coordinate $z_b$, each FMAC takes an input block starting at index $z_b + offset$ and multiplies it by a coefficient. The offset values are $\{0, 1, 2, 3\}$ and $\{5, 6, 7, 8\}$, and corresponding coefficients are $\{c_{z4}, c_{z3}, c_{z2}, c_{z1}\}$ and $\{c_{z1}, c_{z2}, c_{z3}, c_{z4}\}$. The CSL code is provided in Figure 7 and the first 4 steps of this process are illustrated in Figure 6. As can be seen, this step skips offset 4, which would correspond to the multiplication by $c_{z0}$, since that particular computation has already been done as discussed earlier. At the end of this step, the Laplacian is contained in the accumulator buffer.

D. Time integration

Once the Laplacian has been computed, the time iteration step given in Equation 2 can happen. The wavefield
zWF is the wavefield stored in the local memory of a PE

\[ \text{accumulator}_{z_i \leq i < z_i + b} = \text{accumulator}_{z_i \leq i < z_i + b} + zWF_{z_i - 4 \leq i < z_i + b - 4} \times c_4 \]
\[ + zWF_{z_i - 3 \leq i < z_i + b - 3} \times c_3 \]
\[ + zWF_{z_i - 2 \leq i < z_i + b - 2} \times c_2 \]
\[ + zWF_{z_i - 1 \leq i < z_i + b - 1} \times c_1 \]
\[ + zWF_{z_i + 1 \leq i < z_i + b + 1} \times c_{b+1} \]
\[ + zWF_{z_i + 2 \leq i < z_i + b + 2} \times c_2 \]
\[ + zWF_{z_i + 3 \leq i < z_i + b + 3} \times c_3 \]
\[ + zWF_{z_i + 4 \leq i < z_i + b + 4} \times c_4 \]

(a) Operations performed along the Z dimension.

(b) Equivalent CSL code. Each @fmacs instruction takes an output argument and three input arguments. @get_dsd returns a descriptor, corresponding to a view of an array. @increment_dsd_offset allows to offset the array pointed by an existing descriptor.

Fig. 7: Applying the stencil along the Z dimension.

from the previous time step is added to the \text{accumulator} buffer. In reality, this is also done during the stencil computation: as mentioned earlier, a PE receives its own data. Doing so allows to use cycles which would have otherwise been wasted.

The next step is to update the wavefield (per Equation 2), by subtracting the wavefield to the \text{accumulator} buffer (step 3 in Figure 5).

Next, a stimulus, called \textit{source}, needs to be added to a particular cell (with coordinates \((\text{srcX}, \text{srcY}, \text{srcZ})\)) at each time step. The source value at the current time step is added to the wavefield at offset \text{srcZ} on the PE with coordinates \((\text{srcX}, \text{srcY})\).

V. EXPERIMENTAL EVALUATION

In this section, experimental results of Finite Differences running on a Wafer-Scale Engine are presented. The scalability and energy efficiency achieved by Finite Differences on this massively parallel platform are discussed.

A. Experimental Configuration

The experiments are conducted on two platforms: a Cerebras CS-2 equipped with a WSE-2 chip, and a GPU-based platform used as a reference. The CS-2 is Cerebras’ second generation chassis, which uses the 7nm WSE-2 second generation Wafer-Scale Engine. The WSE-2 offers 2.2x more processing elements than the original WSE. The experiments used a fabric of size 755 × 994 out of the total 850,000 processing elements of the WSE-2. The CS-2 is driven by a Linux server on which no computations take place in the context of this work. The WSE-2 platform uses Cerebras SDK 0.3.0 [27].

The GPU-based platform is Cypress from TotalEnergies. It has 4 NVIDIA A100 GPUs, each offering 40 GB of on-device RAM, a 16-core AMD EPYC 7F52 CPU, and 256 GB of main memory. The GPU platform is using CUDA 11.2 and GCC 8.3.1.

Numerical results produced by Finite Differences on WSE-2 are compared to the results produced by Minimod.

B. Weak scaling Experiments

This section discusses scalability results of Finite Differences on a WSE-2 system. In order to characterize the scalability of Finite Differences, the grid dimension is modified along the X and Y dimensions, while the Z dimension (residing in memory) is kept constant to a relevant value for this type of application. The X and Y dimensions are grown up to a size of 755 × 994. Results presented in Table II show the throughput achieved on WSE-2 in Gigacell/s, the wall-clock time required to compute 1,000 time steps on WSE-2, as well as timings on a GPGPU provided as baseline. Timing reported in this section correspond to computations taking place on the device only, be it on GPU or WSE-2.

As can be seen in the table, for all problem sizes, the wall-clock time required on WSE-2 is constant, meaning that Finite Differences scales nearly perfectly on this platform. It is crucial to observe that such a reduction in wall-clock time has a significant impact in practice since this type of computations is repeated hundreds of thousands of times in an industrial context. Finite Differences reaches a throughput of 9872.78 Gcell/s on the largest problem size, which is rarely seen at single system level. This type of throughput is difficult to achieve without using a large number of nodes on distributed-memory supercomputers due to limited strong scalability.

Figure 8 depicts the ratio between the elapsed time achieved by the A100-tuned kernel compared to Finite Differences on WSE-2. As can be seen, when the largest
problem is solved (grid size of $755 \times 994 \times 1000$), a speedup of 228x is achieved. While this number shows great potential, it is understood that using multiple GPUs will likely narrow the gap. However, it is unlikely that such a performance gap can be closed entirely, given the strong scalability issues encountered by this kind of algorithm when using a large number of multi-GPU nodes in HPC clusters ([1], [29]).

Finite Differences shows close to perfect weak scaling on WSE-2. No matter what the grid size is, the run time stays fairly stable. Taking the 200x200 case as a reference, percentages of the ideal weak scaling for various grid sizes are depicted in Figure 9. As can be seen in the plot, Finite Differences systematically reaches over 98% of weak scaling efficiency. This demonstrates how extremely low latency interconnect coupled with local fast memories can be efficiently leveraged by stencil applications relying on a localized communication pattern.

In the next experiment, the sizes of the $X$ and $Y$ dimensions of the grid are fixed to $n_x = 755$ and $n_y = 994$ while the size of the $Z$ dimension $n_z$ is varied from 100 to 1,000. Results presented in Table III show that the throughput increases slightly with $n_z$. This indicates that the implementation gains in efficiency due to larger block sizes $b$ and therefore lower relative overheads. More importantly, it confirms that memory accesses do not limit the performance of the implementation, confirming that it is compute-bound.

![Fig. 8: Comparisons between implementation on WSE-2 and A100 using elapsed time describe in Table II. Fixed $n_z = 1000$.](image)

![Fig. 9: Weak scaling under assumption that PE memory is fully utilized ($n_z = 1000$).](image)

### Table II: Experimental results for 1,000 time steps for various grid sizes with fixed $n_z$.

| $n_x$ | $n_y$ | $n_z$ | Throughput Gcell/s | WSE-2 time [s] | A100 time [s] |
|------|------|------|--------------------|----------------|---------------|
| 200  | 200  | 1000 | 533.64             | 0.0750         | 0.7892        |
| 400  | 400  | 1000 | 2097.60            | 0.0763         | 3.5628        |
| 600  | 600  | 1000 | 4731.53            | 0.0761         | 8.0000        |
| 755  | 500  | 1000 | 4956.17            | 0.0762         | 8.5499        |
| 755  | 600  | 1000 | 5945.40            | 0.0762         | 10.1362       |
| 755  | 900  | 1000 | 8922.08            | 0.0762         | 15.5070       |
| 755  | 990  | 1000 | 9782.14            | 0.0764         | 17.4991       |
| 755  | 994  | 1000 | 9862.78            | 0.0761         | 17.4186       |

### Table III: Experimental results, fixed $n_x \times n_y$ grid dimensions of $755 \times 994$. 100,000 time steps.

| $n_z$ | $b$ | Throughput Gcell/s | WSE-2 time [s] | Scaling |
|------|-----|--------------------|----------------|--------|
| 100  | 100 | 8688.76            | 0.8637         | 1.0000 |
| 200  | 200 | 9303.26            | 1.6133         | 1.8679 |
| 300  | 300 | 9614.15            | 0.3716         | 2.7458 |
| 400  | 400 | 9614.15            | 3.1223         | 3.6151 |
| 500  | 250 | 9786.51            | 3.8342         | 4.4392 |
| 700  | 350 | 9885.04            | 5.3143         | 6.1531 |
| 1000 | 334 | 9936.79            | 7.5524         | 8.7442 |

### C. Profiling data

In the following, various profiling results are provided for Finite Differences, with the objective to provide general insights on WSE-2-based computations.

Using Cerebras’ hardware profiling tool on a $600 \times 600 \times 1000$ grid, the execution of Finite Differences results in an average of 69.6% busy cycles. The 4 PEs at the corners of the grid have 0 busy cycles since they are not doing any computation. There is an average of 11.6% idle cycles caused by memory accesses. As expected, the load is extremely balanced, with a standard deviation of 0.8%. This shows that the hardware is kept busy during the experiment, further confirming the efficiency of the
The power consumption of the CS-2 during a Finite Differences run on a $755 \times 994 \times 1000$ grid is reported in Figure 10. In order to record a sufficient number of samples, the run time is extended by setting the number of time steps to $10,000,000$, leading to a total run time of 754 seconds. The average power consumption during the execution is 22.8 kW, which corresponds to 22 GFLOP/W. Such an energy efficiency is hard to find in the literature for a stencil of this order. In addition to power consumption, Figure 10 also depicts the coolant temperature of the CS-2, which uses a closed-loop water-cooling system. During the execution, the coolant temperature rises very moderately from 23.6°C to a peak of 25.6°C.

Altogether, experiments show that the Finite Differences algorithm presented in this study is able to exploit low-latency distributed memory architectures such as the WSE-2 with very high hardware utilization. The application has near-perfect weak scalability and provides significant speedups over a reference GPGPU implementation.

VI. RooFl ine Model

A roofline model [33] is a synthetic view of how many floating point instructions per cycles can be done. This is the peak compute capacity of the platform. However, no computation can be done without loading data: the number of 32 bit words that can be accessed per cycle will also impact the peak compute rate. In the case of a bandwidth-bound application, the memory will actually

![Fig. 11: Roofline models for WSE-2 and GPGPU-based implementations (in log-log scale) for a $755 \times 994 \times 1000$ grid. Dots represent Finite Differences implementations. WSE-2 (top) has two distinct resources: memory and fabric: the leftmost blue dot corresponds to memory accesses, while the red dot to the right corresponds to fabric accesses. Rooflines are given using the same colors. The kernel is clearly in the compute-bound zone for both memory and fabric accesses. On GPGPU (bottom), red dots and lines correspond to DRAM accesses. L1 cache accesses are depicted in blue. The kernel is clearly in the bandwidth-bound zone.](image)
TABLE IV: Instruction and memory access counts of the Finite Differences implementation on WSE-2

| Operation | FLOP | Mem. traffic | Fabric traffic |
|-----------|------|--------------|----------------|
| 17 FMUL   | 1    | 1/b load, 1 store | 1 load |
| 17 FADD   | 1    | 2 loads, 1 store  | 0 |
| 8 FMA     | 2    | 2 + 1/(b) loads, 1 store | 0 |
| 1 FSUB    | 1    | 2 loads, 1 store  | 0 |

This leads to an arithmetic intensity of 0.11 with respect to memory accesses, and 0.75 with respect to fabric transfers.

On WSE-2, a $755 \times 994 \times 1000$ grid is computed in 0.076s (see Table II). This leads to a flop rate of 670.3 MFLOPs per PE, and an aggregated performance of 503 TFLOPs for the entire grid of PEs used by this problem size. The roofline model of the WSE-2, depicted in Figure 11(top), indicates that Finite Differences is compute bound thanks to the extremely fast local memory. This is quite remarkable, and confirms the weak scaling results given in Section V. The application is communication/memory bound on most architecture, such as the GPU platform used in this study (roofline model depicted in Figure 11(bottom)). Note that different optimizations lead to different arithmetic intensities.

VII. Conclusion

In this work, a Finite Differences algorithm taking advantage of low-latency localized communications and flat memory architecture has been introduced. Localized broadcast patterns are introduced to exchange data between processing elements and fully utilize the interconnect. Experiments show that it is possible to reach near perfect weak scalability on distributed memory architecture such as the WSE-2. On this platform, the implementation of Finite Differences reaches 503 TFLOPs. This is a remarkable throughput for this stencil order on a single node machine. The roofline model introduced in this work confirms that Finite Differences becomes compute bound on the WSE-2. This demonstrates the validity and potential of the approach presented in this work, and demonstrate how different hardware architectures like the WSE-2 can be exploited efficiently by stencil-based applications.

Future efforts include the integration of the ported kernel at the center of more ambitions applications, such as the ones regularly used by seismic modeling experts when taking real-life decisions. Further, given the well established capacity of this hardware architecture for ML-based applications, a hybrid HPC-ML approach will also be investigated.

One interesting consequence of having a relatively compact machine delivering such a high performance level for this type of application is that seismic data processing can happen at the same time it is acquired on the field, which is key when constant monitoring is required. Furthermore, under this scenario, processing capacity can move from data centers closer to where sensors are, namely target edge-HPC.

Acknowledgements

Authors would like to thank Cerebras Systems and TotalEnergies for allowing to share the material. Also, authors would like to acknowledge Grace Ho, Ashay Rane, and Natalia Vassilieva from Cerebras for the contributions, and Ruyichi Sai from Rice U. for fruitful discussions about GPGPU optimized kernels.
execution-cache-memory model,” in Proceedings of the 29th ACM on International Conference on Supercomputing, ser. ICS ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 207–216. [Online]. Available: https://doi.org/10.1145/2751205.2751240

[31] Z. team, “Zig programming language,” https://ziglang.org/, 2018. (Accessed on 03/18/2022).

[32] F. Thaler, S. Moosbrugger, C. Osuna, M. Bianco, H. Vogt, A. Afanasyev, L. Mosimann, O. Führer, T. C. Schulthess, and T. Höfler, “Porting the cosmo weather model to manycore cpus,” in Proceedings of the Platform for Advanced Scientific Computing Conference, ser. PASC ’19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3324989.3325723

[33] S. Williams, A. Waterman, and D. Patterson, “Roofline: An insightful visual performance model for multicore architectures,” Commun. ACM, vol. 52, no. 4, p. 65–76, apr 2009. [Online]. Available: https://doi.org/10.1145/1498765.1498785

[34] D. Wonnacott, “Using time skewing to eliminate idle time due to memory bandwidth and network limitations,” in Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000. IEEE, 2000, pp. 171–180.

[35] K. Zhang, H. Su, P. Zhang, and Y. Dou, “Data layout transformation for stencil computations using arm neon extension,” in 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2020, pp. 180–188.