Computational Mechanics (2019) 64:879–894
https://doi.org/10.1007/s00466-019-01685-4

ORIGINAL PAPER

Acceleration strategies for explicit finite element analysis of metal powder-based additive manufacturing processes using graphical processing units

Mojtaba Mozaffar[1] · Ebot Ndip-Agbor[1] · Stephen Lin[1] · Gregory J. Wagner[1] · Kornel Ehmann[1] · Jian Cao[1]

Received: 23 October 2018 / Accepted: 10 February 2019 / Published online: 1 March 2019

Abstract
Metal powder-based Additive Manufacturing (AM) processes are increasingly used in industry and science due to their unique capability of building complex geometries. However, the immense computational cost associated with AM predictive models hinders the further industrial adoption of these technologies for time-sensitive applications, process design with uncertainties or real-time process control. In this work, a novel approach to accelerate the explicit finite element analysis of the transient heat transfer of AM processes is proposed using Graphical Processing Units. The challenges associated with this approach are enumerated and multiple strategies to overcome each challenge are discussed. The performance of the proposed algorithms is evaluated on multiple test cases. Speed-ups of about 100×–150× compared to an optimized single CPU core implementation for the best strategy were achieved.

Keywords Additive manufacturing · Directed energy deposition · GPU acceleration · Finite element methods · High performance computing

1 Introduction

Metal powder-based additive manufacturing (AM) processes are increasingly used in a variety of industries due to their advantages in geometric complexity and flexibility of their products and consequently, the associated manufacturing time and cost. Nowadays, applications of metal powder-based AM processes go beyond just producing prototypes, but also for manufacturing functional products with complex geometries [1], varying alloy composition [2, 3], and locally-controlled microstructures [4, 5]. Directed Energy Deposition (DED) is a class of AM processes that uses focused heat sources, usually an electron or laser beam, to melt the powders and simultaneously delivers the powder to the focal point of the heat source as the powder delivery nozzle follows the toolpath derived from CAD geometries [6, 7]. Selective Laser Melting (SLM) is another category of AM processes in which a thin layer of powder is delivered to the base plate using a powder delivery system and then the laser is used to melt and fuse the powder [8]. Schematics of these two AM processes, DED and SLM, are demonstrated in Fig. 1.

The uncertainty in predicting the final properties of the products is one of the most critical challenges of AM technologies. Many computational methods have been proposed to address this issue using macro-scale [9, 10], meso-scale [11, 12] or multi-scale modeling [13, 14]. However, a common problem with the existing predictive methods for AM is their enormous computational cost that might take weeks or months of compute time [15], which makes these computational models orders of magnitude slower than the experiment itself and impossible to use in any time-sensitive application such as real-time control or optimization procedures. Therefore, investigating methods to accelerate AM predictive models is vital for overcoming existing barriers and to achieve wider application of AM technologies in industry.

Mojtaba Mozaffar and Ebot Ndip-Agbor have contributed equally to this work.

✉ Jian Cao
jcao@northwestern.edu

[1] Department of Mechanical Engineering, Northwestern University, Evanston, IL 60208, USA

1.1 A review on GPU-accelerated finite element analysis

One approach to overcome the computational burdens associated with the modeling methods is by accelerating AM simulations using parallelization practices on computer clusters or more recently Graphical Processing Units (GPUs). GPUs are traditionally designed to handle computer graphics and their hardware is designed to perform optimally for that task. With the emergence of the General-Purpose GPU (GPGPU) concept, the applications of GPUs have been extended to many science fields and revolutionized computations in finance, bioinformatics, machine learning and computer vision [16].

Finite Element Method (FEM) is an effective tool for simulating the process physics in wide variety of applications including AM [14, 17]. FEM calculations consist of two major tasks: (1) creating a large system of equations based on physics-based partial differential equations on a discretized domain; and (2) solving the system of equations. GPUs can be used to accelerate the process of solving finite element analysis (FEA) systems of equations. Efficient GPU-accelerated libraries, such as THRUST [18], exist that handle the iterative procedure of solving matrix-based equations. Solving sparse systems of equations on GPUs has been extensively investigated [19] and well-developed libraries are publicly available such as cuSPARSE [20]. Recently, commercial FEM software packages, such as ABAQUS, COMSOL, etc., use this technique to boost the performance for their analyses. A benchmark of the acceleration performance of different matrix solvers for the simulation of polymer actuator's electromechanical response is developed in [21]. An implicit simulation of an automobile battery's thermal runaway is accelerated using Thrust and Paralution [22] libraries as equation solvers with speed-ups of up to 50 in [23].

An alternative approach is to perform both tasks (i.e., creating the FEM systems of equation and solving them) on the GPU. A fundamental investigation of this method is presented in [24] for an FEM simulation which solves steady-state heat equations. Different work distributions and memory arrangements for the implementation are considered resulting in speed-up of up to 30× depending on the element order and simulation size (i.e., number of elements in the simulation). A high-level domain-specific language was developed for the implementation of FEM simulations on both CPUs and GPUs in [25, 26]. Using the developed platform, called the Unified Form Language (UFL), they investigated the memory storage and access patterns that led to optimal performances in CPU and GPU implementations. Furthermore, they introduced the Local Matrix Approach (LMA) as an alternative assembly algorithm to eliminate the necessity for operations known as atomic operations. In [27–29], the matrix generation method is divided into three consecutive tasks of: (1) numerical integration, (2) assembly in the coordinate (COO) format, and (3) conversion into Compressed Row Storage (CRS) [30] format. The authors used GPU computing for simulating 9-pole microwave electromagnetic responses by distributing the GPU work based on each FEM integration point. Using a sparse solver with the Conjugate Gradient Method (CGM) and preconditioners, the authors achieved a 81× speed-up of the simulation. Global memory accesses and calculations are interleaved to achieve 100 billion floating-point operations per second in [31, 32]. A mapping between elements and integration points was proposed to eliminate the reduction operations. In Ref. [33], the authors discussed the existing works and potentials of GPU computing for the structural analysis components including model conversion, meshing, solving the system of equations, and visualizing the results.

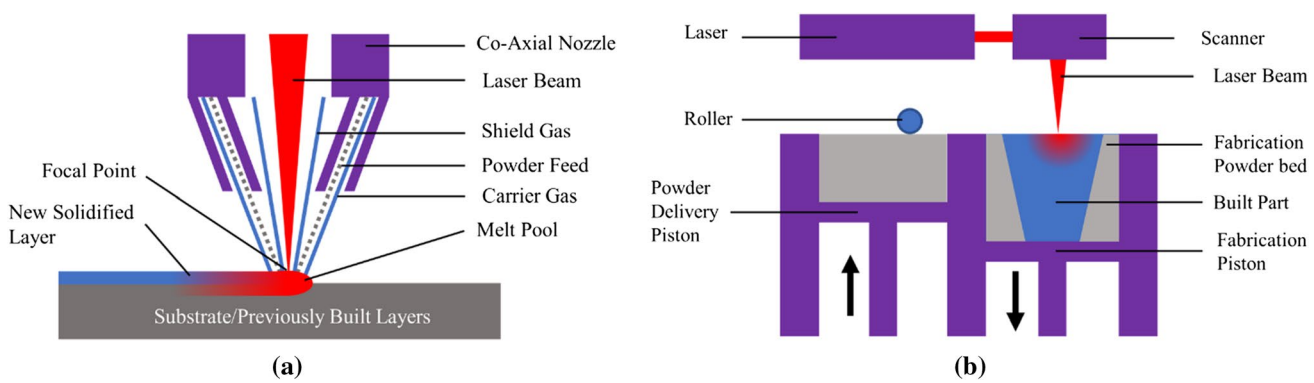


Fig. 1 Schematic of two AM processes; **a** DED process with coaxial nozzle to deliver powder to the focal point of a laser and **b** SLM process that uses a roller to spread a thin layer of powder before melting the layer [53, 54]

1.2 Scope of this paper

Despite existing profound researches on acceleration of FEA solvers, the methodologies discussed above do not provide a solution for the implementation of an explicit FEA on a problem with advanced evolving boundary conditions, which is needed for simulating a metal powder-based AM process. An explicit FEA solver does not require iterative solutions of large matrices and thus requires less computation than an implicit solver for a single time step, resulting in better solution efficiency for small time-step sizes. In many cases, a high resolution time-step is vital for metal-based AM considering that melting and solidification happen very fast due to the highly localized laser heating. This paper proposes a novel way of accelerating explicit FEA on GPUs and demonstrates that well-crafted executive strategies, data structures, and algorithms can achieve remarkable speed-ups for explicit FEA. Due to the importance of the thermal history of metal powder-based AM processes in the formation of microstructures, residual stress, and distortions [34–36] of the additive manufactured products, this study focuses on establishing an accelerated transient heat transfer FEA.

A summary of the finite element formulations for the transient heat transfer and the GPU execution model is presented in Sects. 2 and 3 respectively. Alternative strategies for matrix assembly and heat flux calculations are proposed in Sect. 4 along with optimization methods to maximize GPU memory bandwidth and occupancy. The performance of the acceleration strategies on multiple test cases, the reasons behind the performance of each alternative and the accuracy of the calculations is verified and discussed in Sect. 5 followed by conclusions in Sect. 6.

2 Finite element formulation for transient heat transfer

This section introduces a summary of the underlying FEA formulation for thermal analysis of AM in order to establish the equations that will be used in the rest of the work. Here, only the key formulations are highlighted while the detailed mathematical steps can be found in [37–39]. First, the weak form of the transient heat equation will be derived from the governing equations and the boundary conditions. This weak form will then be discretized using the Galerkin method for each element. Then the Gauss quadrature and explicit time integration schemes will be used to solve the global system of equations assembled from the local system of each element.

The governing equation for the transient heat transfer that is to be solved can be written as [37]:

$$\rho c_p \frac{\partial T}{\partial t} - \nabla \cdot (k \cdot \nabla T) - s = 0 \tag{1}$$

where ρ is the material density, c_p is the specific heat capacity, T is temperature, t is time, k is the material conductivity, and s is the heat generation rate per unit volume. The following boundary conditions are considered in this work:

$$(1)\ \text{Dirichlet}\ \ T = T_1(x, y, z)\ \text{on}\ \Gamma_1 \tag{2}$$

$$(2)\ \text{Neumann}\ \ q = -q_s\ \text{on}\ \Gamma_2 \tag{3}$$

$$(3)\ \text{Convection}\ \ q = -h\left(T - T_{amb}\right)\ \text{on}\ \Gamma_3 \tag{4}$$

$$(4)\ \text{Radiation}\ \ q = -\varepsilon\sigma\left(T^4 - T_{amb}^4\right)\ \text{on}\ \Gamma_4 \tag{5}$$

where q_s is the external heat flux, h is the convection coefficient, T_{amb} is the ambient temperature, ε is the surface emissivity constant, σ is the Stefan-Boltzmann constant, and Γ_1, Γ_2, Γ_3 and Γ_4 are sets of surfaces on which each of these boundary conditions are applied on. Note that the surface sets Γ_2, Γ_3 and Γ_4 are not exclusive which means multiple boundary conditions can be assigned to a surface.

By multiplying Eq. (1) by a differentiable weight function $\omega(x)$ with $\omega(x) = 0$ on Γ_1 and using integration by parts, the weak form of the transient heat transfer equation is obtained as follows:

$$\begin{aligned} &\int_\Omega \rho c_p \tfrac{\partial T}{\partial t}\omega dV + \int_\Omega (\nabla\omega)\cdot(k\nabla T)dV - \int_\Omega s\omega dV + \int_{\Gamma_2}\left(q_s\cdot n\right)\omega dA \\ &\quad + \int_{\Gamma_3} h\left(T - T_{amb}\right)\omega dA + \int_{\Gamma_4}\varepsilon\sigma\left(T^4 - T_{amb}^4\right)\omega dA = 0 \\ &\quad \forall\omega(x) with \omega(x) = 0 on \Gamma_1 \end{aligned} \tag{6}$$

where $T = T_1(x, y, z)$ on Γ_1.

The weight function and temperature field are discretized using standard finite element shape functions and their derivatives, which result in:

$$T^e = \left[N^e\right]\left[L^e\right]\{T\}, \quad \omega^e = \left[N^e\right]\left[L^e\right]\{\omega\} \tag{7}$$

$$\nabla T^e = \left[B^e\right]\left[L^e\right]\{T\}, \quad \nabla\omega^e = \left[B^e\right]\left[L^e\right]\{\omega\} \tag{8}$$

where $[N^e]$ is the matrix of shape functions, $[B^e]$ is its derivative for each element,$[L^e]$ is the gather matrix, $\{T\}$ is the vector of temperatures at the nodes, and $\{\omega\}$ is the vector of weight function values. The discretized form of Eq. (6) can be written as [37]:

$$
\begin{aligned}
&\underbrace{\left(\sum_{e=1}^{n_{el}}[L^e]^T\underbrace{\int_{\Omega^e}\left(\rho^e c_p^e\left[N^e\right]^T\left[N^e\right]\right)dV[L^e]}_{\mathrm{M^e}}\right)}_{[\mathrm{M}]}\frac{\partial\{T\}}{\partial t}\\
&+\underbrace{\left(\sum_{e=1}^{n_{el}}[L^e]^T\underbrace{\int_{\Omega^e}\left(k^e\left[B^e\right]^T\left[B^e\right]\right)dV[L^e]}_{[\mathrm{K}^e]}\right)}_{[\mathrm{K}]}\{T\}-\underbrace{\underbrace{\sum_{e=1}^{n_{el}}[L^e]^T\int_{\Omega^e}\left(s^e\left[N^e\right]^T\right)dV}_{\{\mathrm{R_G^e}\}}}_{\{\mathrm{R_G}\}}\\
&+\underbrace{\underbrace{\sum_{e=1}^{n_{el}}[L^e]^T\int_{\Gamma_2^e}\left(q_s^e\left[N^e\right]^T\right)dA}_{\mathrm{R_F^e}}}_{\{\mathrm{R_F}\}}\\
&+\underbrace{\underbrace{\sum_{e=1}^{n_{el}}[L^e]^T\int_{\Gamma_3^e}\left(h^e\left[N^e\right]^T\left(\left[N^e\right]\left[L^e\right]\{T\}-\left\{T_{amb}^e\right\}\right)\right)dA}_{\mathrm{R_C^e}}}_{\{\mathrm{R_C}\}}\\
&+\underbrace{\underbrace{\sum_{e=1}^{n_{el}}[L^e]^T\int_{\Gamma_4^e}\left(\varepsilon\sigma\left[N^e\right]^T\left(\left(\left[N^e\right]\left[L^e\right]\{T\}\right)^{\circ 4}-\left\{T_{amb}^e\right\}^{\circ 4}\right)\right)dA}_{\mathrm{R_R^e}}}_{\{\mathrm{R_R}\}}=0
\end{aligned}
\tag{9}
$$

where n_{el} is the number of elements in the domain, superscript 'e' indicates that the parameter is associated with element e, $^\circ$ is the element-wise power operation, [M] is the capacitance matrix, [K] is the conduction matrix, $\{\mathrm{R_G}\}$ is the internal heat vector, $\{\mathrm{R_F}\}$ is the external flux vector, $\{\mathrm{R_C}\}$ is the convection vector, and $\{\mathrm{R_R}\}$ is the radiation vector. This equation shows that FEA matrices and vectors can be calculated separately for each element and then assembled into global variables for solving the weak form equation.

The Gauss quadrature method [40] is used to simplify the numerical evaluation of integrals in Eq. (9). To do so, elements need to be transformed into an isoparametric coordinate system, then the integrals can be calculated by summing up each integrand over the integration points. By applying this transformation, the elemental matrices and vectors defined in Eq. (9) can be reformulated as:

$$\left[\mathrm{M^e}\right]=\sum_{m=1}^{n_G}\left(\rho c_p\left[N_m^e\right]^T\left[N_m^e\right]\omega_m|J_m|\right) \tag{10}$$

$$\left[\mathrm{K^e}\right]=\sum_{m=1}^{n_G}\left(k\left[B_m^e\right]^T\left[B_m^e\right]\omega_m|J_m|\right) \tag{11}$$

$$\left\{\mathrm{R_G^e}\right\}=\sum_{m=1}^{n_G}\left(s\left[N_m^e\right]^T\omega_m|J_m|\right) \tag{12}$$

$$\left\{\mathrm{R_F^e}\right\}=\sum_{n=1}^{n_F}\left(q_s\left[N_n^e\right]^T\omega_n|J_n|\right) \tag{13}$$

$$\left\{\mathrm{R_C^e}\right\}=\sum_{n=1}^{n_F}\left(h\left[N_n^e\right]^T\left(\left[N_m^e\right]\left\{T_n\right\}-\left\{T_{amb}\right\}\right)\omega_n|J_n|\right) \tag{14}$$

$$\left\{\mathrm{R_R^e}\right\}=\sum_{n=1}^{n_F}\left(\varepsilon\sigma\left[N_n^e\right]^T\left(\left\{\left[N_m^e\right]\left\{T_n\right\}\right\}^{\circ 4}-\left\{T_{amb}\right\}^{\circ 4}\right)\omega_n|J_n|\right) \tag{15}$$

where $n_G=8$ and $n_F=4$ are the number of Gauss quadrature integration points for 8-node hexahedron elements used in

this work, $\omega_m = \omega_n = 1$ are the weights of integration points, $\left[N_m^e\right]$ and $\left[B_m^e\right]$ are the shape function and its derivative for 8-node elements in the isoparametric coordinate system, and $\left|J_m\right|$ is the determinant of the Jacobian matrix for the transformation from the Cartesian to the isoparametric coordinate system for 8-node hexahedral elements. $\left[N_n^e\right]$ and $\left|J_n\right|$ are the isoparametric shape functions and the determinant of the Jacobian matrix for 4-node quadrilateral surfaces.

After calculating each of the local matrices and vectors for each element (e.g., $\left[\mathrm{M}^{\mathrm{e}}\right]$), the contribution of each connected node to the global matrices (e.g., [M] in Eq. 9), i.e., FEM assembly, should be performed. While traditionally assembly is done using gather matrices (Eqs. 7–8), it is more memory efficient to store the global IDs and coordinates of all connected nodes in an object-oriented manner and use them to write the contributions to the associated memory address in the global matrices. Furthermore, the multiplication of the conduction matrix [K] with the nodal temperature $\{T\}$ can be moved ahead of the assembly of the global conduction matrix. Using this approach, one can calculate the local conduction flux as $\{C^e\} = \left[\mathrm{K}^{\mathrm{e}}\right]\{T^e\}$ and then assemble it into the global conduction flux vector, $\{\mathrm{C}\}$. This significantly reduces memory consumption as it avoids storing the $n_n \times n_n$ global conduction matrix and, instead, stores the conduction flux vector of length of n_n, where n_n is the number of nodes in the simulation.

A forward Euler time integration scheme is used to approximate the temperature derivative as presented hereunder:

$$\begin{aligned}[\mathrm{M}]\left(\tfrac{1}{\Delta t}\left(\left\{T^{n+1}\right\} - \{T^n\}\right)\right) &= \left\{\mathrm{R_G}\right\} - \left\{\mathrm{R_F}\right\} - \left\{\mathrm{R_C}\right\} - \left\{\mathrm{R_R}\right\} - \{\mathrm{C}\} \\ \left\{T^{n+1}\right\} &= \{T^n\} + \Delta t[\mathrm{M}]^{-1}\left[\left\{\mathrm{R_G}\right\} - \left\{\mathrm{R_F}\right\} - \left\{\mathrm{R_C}\right\} - \left\{\mathrm{R_R}\right\} - \{\mathrm{C}\}\right]\end{aligned} \tag{16}$$

where Δt is the time step and $\left\{T^{n+1}\right\}$ and $\{T^n\}$ are nodal temperatures at time steps $n+1$ and n, respectively. The summation of $\left\{\mathrm{R_F}\right\}$, $\left\{\mathrm{R_C}\right\}$, and $\left\{\mathrm{R_R}\right\}$ is referred to as the external flux while $[\mathrm{K}]\{T^n\}$ as the conduction flux. Furthermore, $\left\{\mathrm{R_G}\right\} = 0$ for AM processes since elements do not generate internal heat.

A common approach to solve Eq. (16) more efficiently is to convert [M] into a diagonal matrix. This can be done by considering the summation of each row as the diagonal value [41]. This operation, also called mass lumping, not only will eliminate the need to calculate the inverse of a large matrix, but also makes the calculations for the temperature of each node independent from other nodes. The explicit thermal analysis of metal powder-based AM with a forward Euler integration time scheme is considered as an acceptable numerical approach since it has been shown to provide many useful physics-informed features for computing microstructural evolution and mechanical properties of AM-built parts [14, 39, 42, 43].

3 Massively Parallel computing with CUDA: execution and memory model

While CPUs consist of a few processing cores that have been optimized for sequential and complex processing, GPUs consist of thousands of smaller cores designed for highly parallel tasks. CUDA is an API (Application Programming Interface) provided by NVIDIA (inventor of the GPU) that enables developers to manage devices and memory allocations on both CPUs and GPUs to efficiently solve complex problems. CUDA can be used through compiler directives, CUDA-enabled

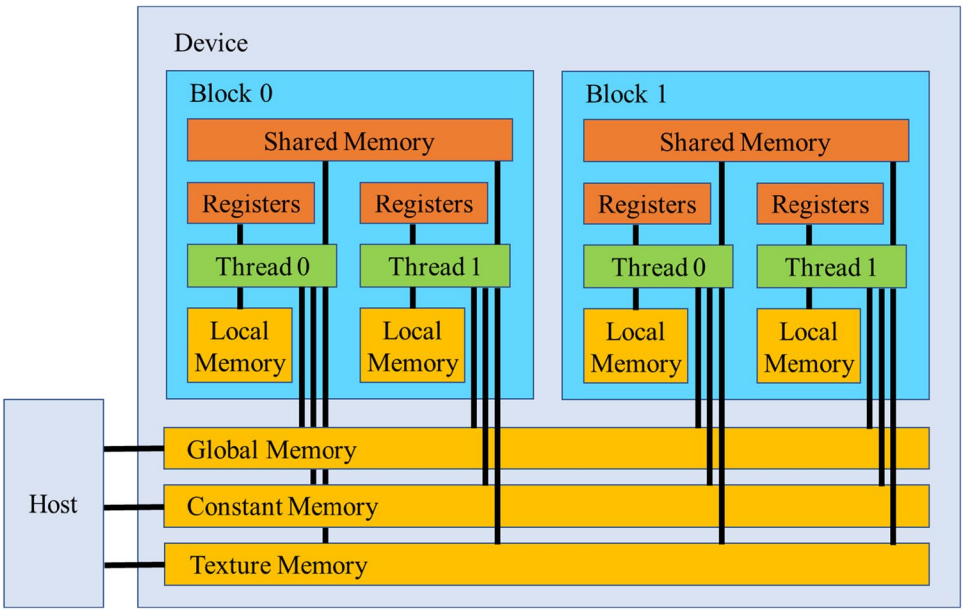


Fig. 2 CUDA hierarchical memory model; device (GPU) can communicate with host (CPU) through global, constant, and texture memories, accessible to all threads. Registers and shared memory are low latency memories exclusively visible to a thread and a block respectively

libraries, and multiple programming languages such as Fortran, Python, C, C++. The present work is developed with the CUDA C/C++ compiler, which is included in the NVIDIA CUDA Development Kit 9.

GPUs are mainly made from multiple Streaming Multiprocessors (SMs) with the key components of computing cores, logical and memory operational units, scheduler, and on-chip memories. Each SM can execute hundreds of threads at the same time based on the resources available to them. Kernels are launched in a user-defined grid of thread blocks, where each block contains up to 1024 threads in recent GPUs. Once a kernel is launched, its thread blocks will be scheduled to run on different SMs and will remain on the SM scheduler until its execution completes. SMs execute thread blocks in groups of 32 threads called warps. Ideally, all the threads in a warp concurrently execute memory and logical operations, which would lead to the most efficient utilization of GPU resources [44].

GPUs use a programmable hierarchical memory structure that allows developers to optimize the performance of memory operations using multiple types of memories with different capacity, latency, and bandwidth. As shown in Fig. 2, the main memory types in order of decreasing bandwidth are: registers, shared memory, texture memory, local memory, constant memory, and global memory.

Global memory is connected to the CPU memory via PCI Express ports. On the latest GPUs the size of global memory can go up to 24 GB with a memory bandwidth of 900 GB/s [45] which speeds up data transfer operations. Each SM has on-chip shared memory with the capacity of 64 KB on recent NVIDIA GPUs, which allows even faster access than global memory. Each thread that runs on a CUDA core has limited dedicated memory called registers with low latency and high bandwidth. Constant memory is statically defined and resides outside of any kernel. This memory has a dedicated cache per SM and is visible to all the threads. Local memory is physically the same memory as global memory with high latency and is specifically designed for storing variables in threads that cannot fit into registers.

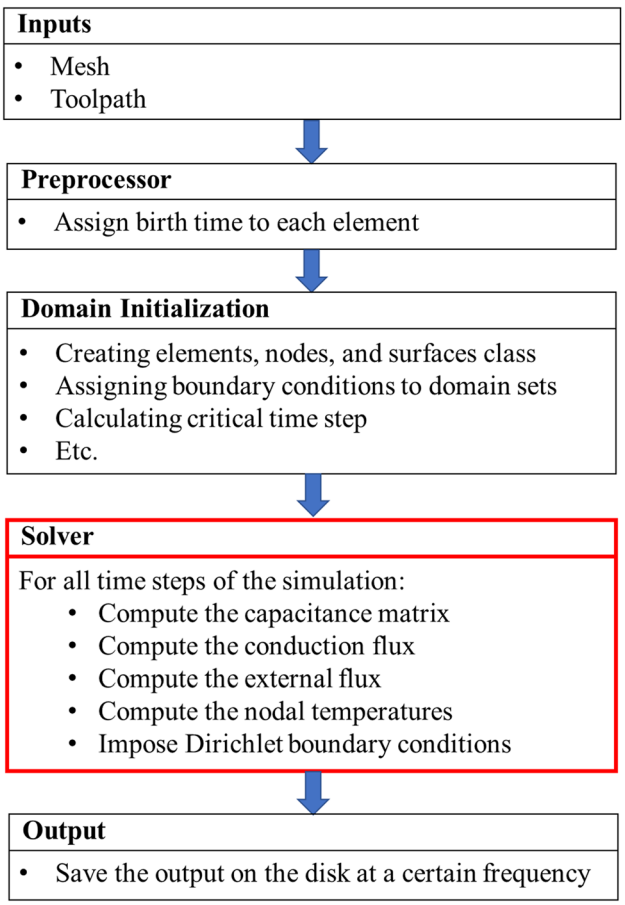


Fig. 3 Computational FEA framework for the thermal analysis of the AM process; the routine includes preprocessing, domain initialization, the solver, and the outputting steps with the solver step as the most computationally expensive one

4 Acceleration strategies

As seen in Sect. 0, the FEA formulation for the thermal analysis of AM processes discretizes the domain into a large number of elements and performs very similar calculations on them, which makes this operation well-matched with the massively parallel architecture of GPUs. However, some operations for this formulation are not inherently parallel and different calculations need to be done on subsets of elements, nodes, and surfaces, especially considering the dynamic nature of this simulation including birth and death of entities and its advanced boundary conditions, which impose serious disadvantages for GPU computing. In the following sections, first the computational framework for the thermal analysis of AM processes is introduced and then the challenges associated with the GPU acceleration of the AM processes are discussed; finally, acceleration strategies for work distribution, memory management, and optimized data-structures are presented to avoid or mitigate the existing challenges.

4.1 FEA computational framework

The overall routine for the thermal analysis of AM processes is demonstrated in Fig. 3, including preprocessing, domain initialization, solver, and outputting steps. The analysis starts with preprocessing the mesh and toolpath files to determine the birth time for each element in the mesh file. Domain initialization creates element, node and surfaces and fills them with information necessary for the simulation. This step is

Thread 0
Thread 1

Node ID	Global Capacitance
1	E1, E2
2	E1, E3, E4,E5
3	E1, E2, E3, E5
4	E2, E3, E4, E6
5	E3, E6, E8
6	E4

(a)

Node ID	Elemental Contributions to Global Capacitance			
1	E1	E2	0	0
2	E1	E3	E4	E5
3	E1	E2	E3	E5
4	E2	E3	E4	E6
5	E3	E6	E8	0
6	E4	0	0	0

(b)

Fig. 4 Assembly strategies for global capacitance; **a** direct assembly to global capacitance may cause a race condition, while **b** the node-element data structure considers separate placeholders for contribution of each element to a node solves this issue

also responsible for assigning the aforementioned boundary conditions to different sections of the mesh and calculating the critical explicit time step to ensure the stability of the simulation. Explicit time stepping is performed during the solver step. In this step, each element is numerically integrated and their contributions are assembled into global matrixes and vectors. In the next time step temperatures for all the nodes are calculated as formulated in Eq. (16). Finally, it is necessary to frequently save the results of the simulation into files on the disk.

Although there is a potential in accelerating the preprocessing and domain initialization steps, these steps run only once for each simulation and their execution time is relatively negligible compared to the time needed for the calculation of the solver steps which repeat for all time steps of the simulation. Therefore, in this work the preprocessing and domain initialization steps are executed on a CPU and the focus of the GPU acceleration is put on the solver step and its efficient interaction with the outputting step.

4.2 Assembly strategy to avoid race condition

An important step in calculating capacitance, conductivity, and flux matrices is assembly, where the contribution of all elements connected to a node are calculated and summed up to the node's global variable. However, considering that these elemental calculations are done concurrently, it is possible that multiple elements at the same time access the global variable of a shared node between them and cause a race condition. Race conditions occur when more than one thread attempts to write to a memory location at the same time. In this event, the output is undetermined. Three assembly strategies are considered as alternatives to overcome this hurdle. While there are similarities between the methods investigated in this work with concepts proposed by Cecka et al. [24] and Markall et al. [25], this investigation is unique because of the factors such as the dynamic birth and death of elements and surfaces in the FEA analysis of AM processes, which are not considered in the existing works. As will be demonstrated below these factors significantly affect the outcome of the simulations. Furthermore, recent advances in both hardware and software capabilities of GPU computing encourage a re-evaluation of the assumptions made in previous studies.

As the first strategy—Node-element structure—the data structure shown in Fig. 4 is considered in this work by assigning a separate memory location for each element connected to a node. Using this data structure, each element will write its contribution to a unique memory and avoid the race condition. To use this structure efficiently, the column index of the contribution of different elements to their shared node need to be pre-assigned in the domain initialization step. For example, column index of the contributions of the elements E3, E6 and E8 to node 5 is 0, 1 and 2 respectively in Fig. 4b. The column indexes, which need to be unique for elemental contributions to shared nodes, are calculated using Algorithm 1 and stored in global memory. Note that a separate kernel is used for calculating the summation of the already calculated contributions in the node-element data structure for each node.

As can be seen from Fig. 4b, the node-element structure contains unused spaces, which may cause inefficient use of global memory especially if the mesh structure contains nodes that are connected to a large number of elements. This problem associated with unequal connected elements can be solved by packing the subarrays for each global node into rows, or bins, of another array using a bin packing algorithm [46], or a more efficient packing algorithm such as the Largest-Processing-Time (LPT) [47]. However, since the current work is focused on the acceleration of the process, this inefficiency in global memory storage is not investigated.

Algorithm 1: Arrange column indexes for node-element data structure

1. Initialize $Col\_ind(n_e, n_i)$ to a zero matrix, where n_e and n_i are the number of elements and nodes in an element, respectively, and Col_ind is the matrix containing unique column indexes for elemental contributions to shared nodes
2. Initialize $I(n_g)$ to a zero matrix, where n_g is the number of global nodes
3. Loop over all the elements e
    a. nodes ← get nodes in the element e
    b. Loop over the nodes n
        i. j ← get the global index of the node n
        ii. $Col\_ind(e, n) = I(j)$
        iii. Increment $I(j)$ by one to generate a unique ID next time that an element accesses this node
    c. End loop over the nodes
4. End loop over the elements

For the second assembly strategy—Coloring—the global matrix calculations are divided into smaller subtasks, where each subtask is responsible for the calculations of a predefined group of elements. By choosing the predefined groups in a way that no two elements in the same group have any shared nodes and performing the calculations for each group sequentially, one ensures that contributions of the elements to any node are written to their allocated memories at different times and, therefore, avoid the race condition. This strategy is known as coloring the mesh as one can arrange the groups by assigning different color codes to the elements in a way that no two adjacent elements have the same color. A disadvantage of this approach is that arranging the colors adds a significant overhead to the domain initialization step of the analysis.

The third approach—Atomic—is to use Atomic operations to perform the assembly. Atomic operations are special types of read-modify-write actions that allow memory addresses to be accessed by only one thread at a time [45]. In the literature, the other alternatives provide better acceleration than Atomic operations because of the steep performance cost associated with them. However, recent advances in GPUs with Compute Capability of 3× or higher improved the performance of Atomic operations. Thus, it is important to reinvestigate the use of Atomic operations for the assembly.

4.3 Mitigating warp divergence

Another issue that might severely affect the performance of explicit FEA for DED processes is warp divergence. Unlike CPUs that use complex branch prediction, GPUs have a simpler flow control mechanism that tries to execute the exact same instructions for all threads in a warp simultaneously. Executing instructions such as an if-else condition causes the *if* block and the *else* block to be performed in sequence instead of in parallel, which adversely affects computational performance.

Explicit FEA for DED processes inherently cause many conditional statements. One major source of the conditional statements is the fact that elements and surfaces might get born or die as the time of the simulation goes on due to the nature of the process that deposits new elements while building the part. To mitigate this issue, both elements and surfaces are sorted based on their birth time in the domain initialization step. By adjusting kernel execution boundaries one can control the range of birth time associated with elements and surfaces that are accessible by each kernel. Considering that the elements stay activated after their birth time, the kernel execution boundaries are dynamically updated in each time step to only perform the kernels on the active elements. The kernel boundary update is implemented using a binary search to efficiently locate the last active element for any time step.

Surfaces can die after their birth time in the FEA for DED processes because an active surface for flux calculations should be on the exterior of the build and the exterior changes dynamically in the building process. Therefore, the dynamic kernel boundary update method only limits the range of active surfaces, but it is not enough to exclusively select active surfaces. Two strategies are considered to be combined with the dynamic boundary update for surface flux calculations. The first strategy is to minimize the branch divergence by executing the flux calculations on all the surfaces in the execution boundary and canceling the effect of inactive surfaces on $\{R_F\}$, $\{R_C\}$ and $\{R_R\}$ in Eq. (16) using a switch. The switch is an integer variable which has a value of 1 for active surfaces and 0 for inactive ones. Using a switch limits warp branching to only a single operation. The second strategy is to prevent the GPU from performing the calculations for inactive surfaces by using a conditional statement at the beginning of the kernel. This strategy would

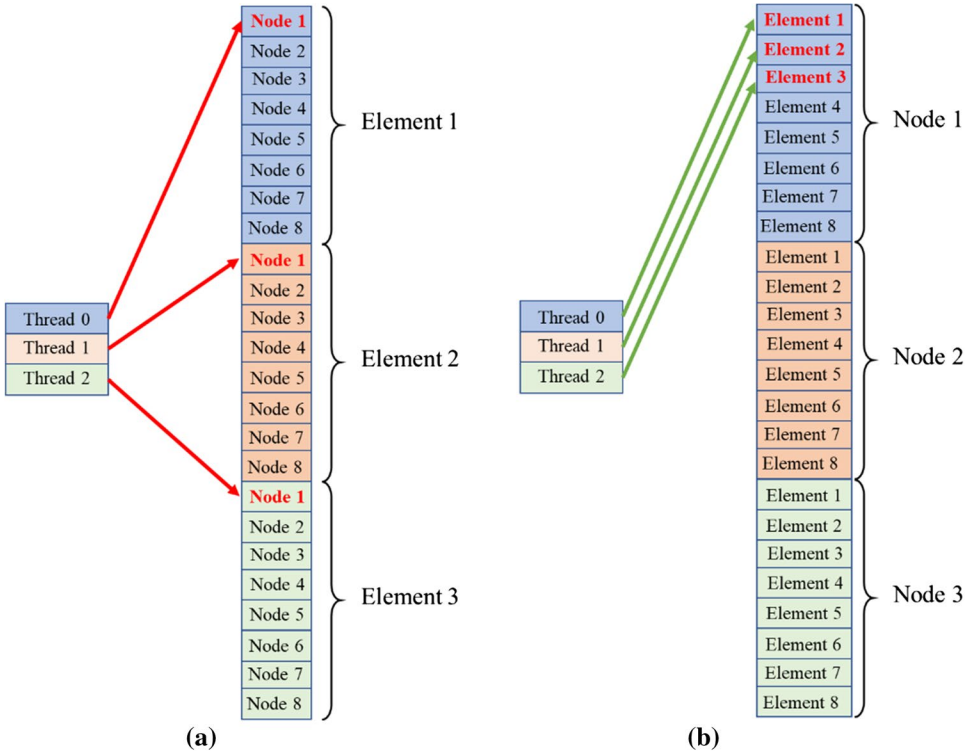


Fig. 5 Global memory access pattern; **a** an uncoalesced access pattern is caused when threads access memories of nodal data for different elements, and **b** a coalesced memory access pattern achieved by rearranging data based on their kernel access

result in fewer computations while inducing a severe branch divergence to warps.

Note that for element-based calculations, it is assumed that the time and location of the elements to be generated are predefined by the toolpath and the elements do not die after their birth considering the nature of AM processes. If this assumption does not hold true (for instance due to mesh adaptivity with dynamic birth and death of elements during the simulation), the balance between branch divergence and redundant calculation penalties determines the optimal performance of element-based calculations similar to surface flux calculations.

Another source of warp divergence is related to different boundary conditions associated with different subsets of the domain. While the Dirichlet boundary condition is usually applied after calculation of temperatures as shown in Fig. 3,

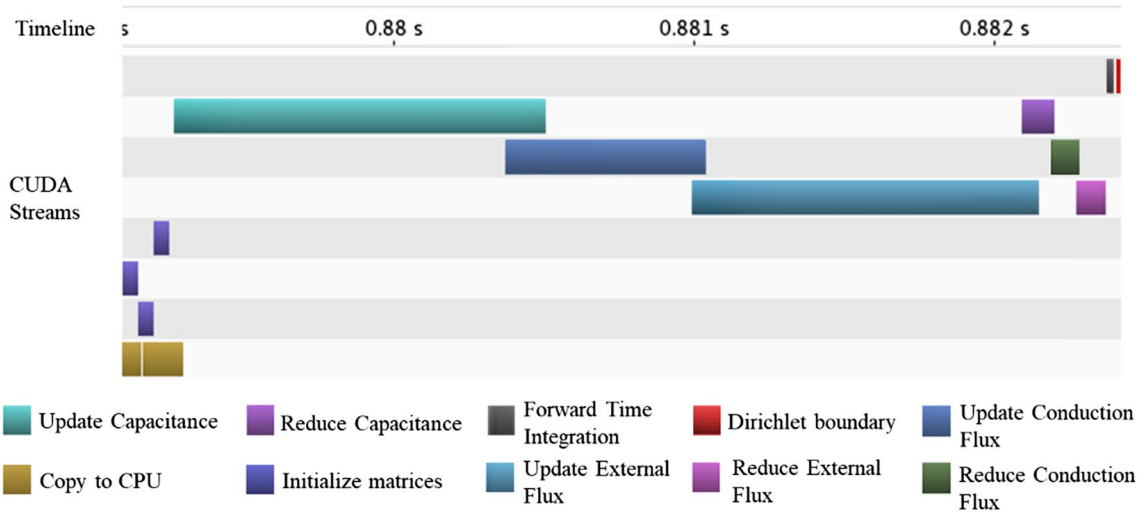


Fig. 6 Visualization of asynchronous kernel execution using NVIDIA visual profiler; rows represent different CUDA streams and each color represents a kernel execution (Color figure online)

surface flux boundary conditions (i.e., external laser flux, convection and radiation) can be applied on different sets of surfaces. Considering that the majority of the flux operations, such as Jacobian and shape function calculations, are similar for the three types, fluxes are calculated in active surfaces for all three types of boundary conditions and the effect of each boundary condition is controlled by using precomputed switches to avoid warp divergence.

4.4 Further optimization considerations

Further optimizations applied to this work will be discussed in this section. Most of the device data reside in the global memory and an efficient access to this memory is essential for achieving high bandwidth in data transactions and proper kernel performance. In the CUDA execution model, memory operations are issued per warp. The most efficient access pattern to the global memory of GPUs is aligned coalesced access. In this access pattern, 32 threads in a warp access a contiguous section of memory starting from an even multiple of the cache size [44]. In this case, a single memory load/write operation is needed for all the threads in a warp leading to a 100% bus utilization. Using uncoalesced or non-aligned data structures would cause the same memory load/write to be done in multiple separate operations.

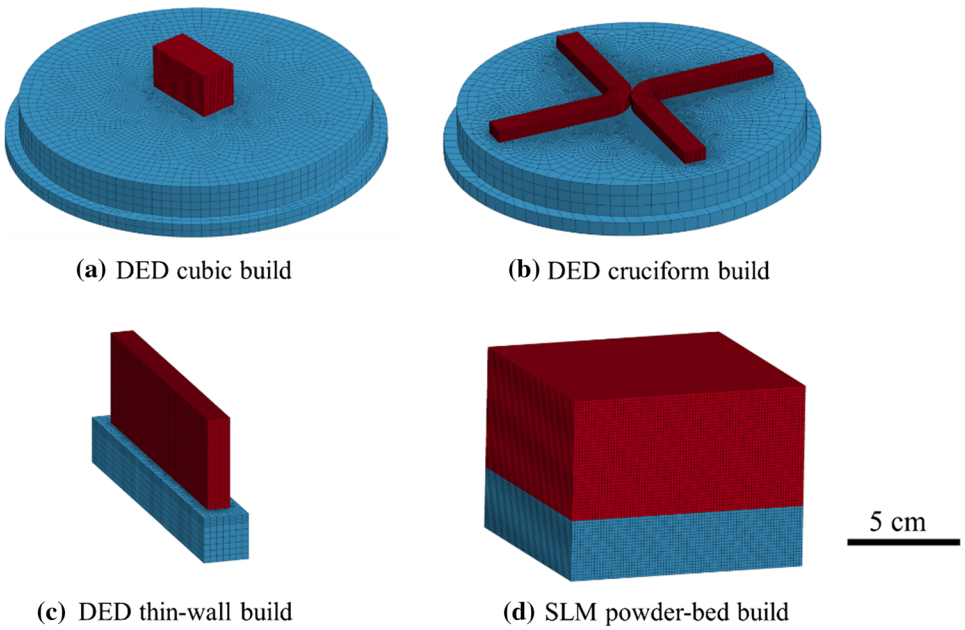
The data associated with different nodes of an element are normally stored next to each other as demonstrated in Fig. 5a. While executing kernels on elements, the GPU warp scheduler will try to execute a single task, for example, calculating the capacitance matrix, for hundreds of elements at the same time. Therefore, all threads in a warp will run memory access for the same index node of all the elements together. This access pattern will cause a significant efficiency penalty due to uncoalesced access.

To maximize the efficiency of global memory reads and writes, data is rearranged in the domain initialization step to access elemental matrices and vectors in a coalesced manner as depicted in Fig. 5b. A similar rearrangement is applied for all element, node, and surface global variables such as the nodal coordinates, the connectivity matrices, the element capacitance and conductivity matrices, and nodal temperatures to ensure efficient global memory transactions. When using the coloring assembly strategy this rearrangement should be done for each color separately.

GPU memory hierarchy architecture allows programmers to use memory types that are most suitable for a given task based on memory latency, bandwidth and capacity in different sections of the code. The GPU constant memory and shared memory are used in the present work to decrease the number of registers used in each kernel and avoid spilling

Table 1 Summary of simulation parameters for the test samples

	Cubic	Cruciform	Thin-wall	Powder-bed
Number of elements	84,346	205,618	193,944	384,000
Number of nodes	93,748	232,447	210,000	400,221
Minimum time step	9.78e−4 s	1.96e−3 s	1.69e−2 s	1.38e−3 s
Build time	1195 s	3007 s	2741 s	42 s
Material	SS316L	SS316L	Ti-6Al-4 V	SS316L
Laser power	1050 W	1050 W	1500 W	120 W
Hatch Spacing	1.1 mm	1.1 mm	1.9 mm	0.5 mm



(a) DED cubic build **(b)** DED cruciform build

(c) DED thin-wall build **(d)** SLM powder-bed build

Fig. 7 Geometries and meshes of the test samples where blue meshes represent the substrate and red meshes represent the build for **a** DED cubic, **b** DED cruciform, **c** DED thin-wall, and **d** SLM powder-bed geometries

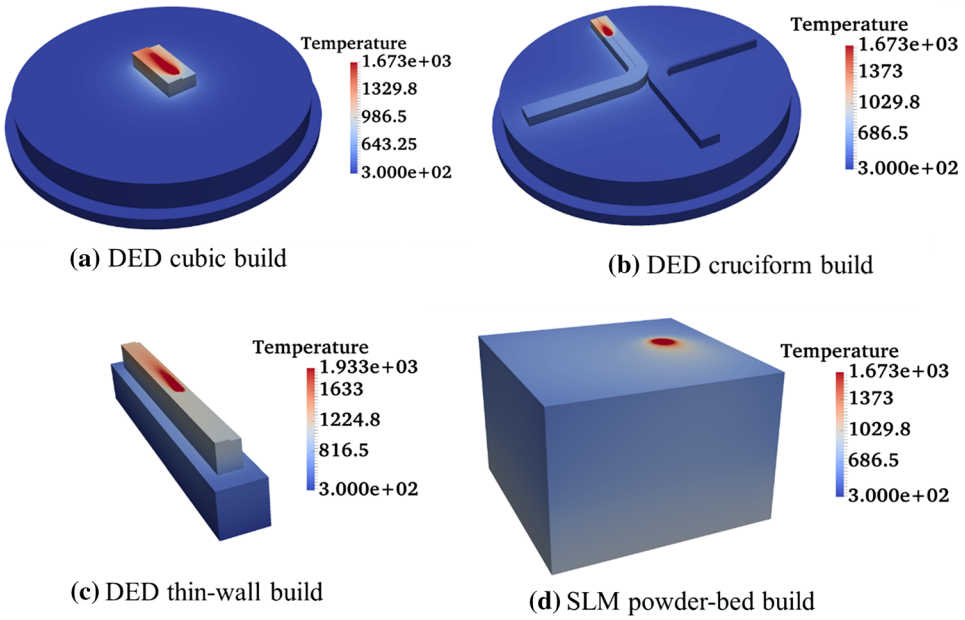


Fig. 8 Visualization of the test simulation outputs for a DED cubic, **b** DED cruciform, **c** DED thin-wall, and **d** SLM powder-bed builds

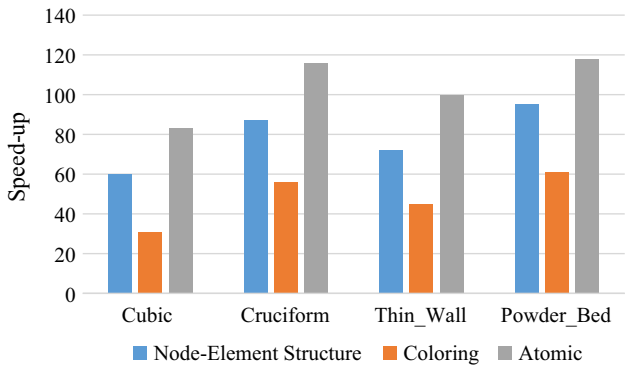


Fig. 9 Acceleration results of the assembly strategies for test samples

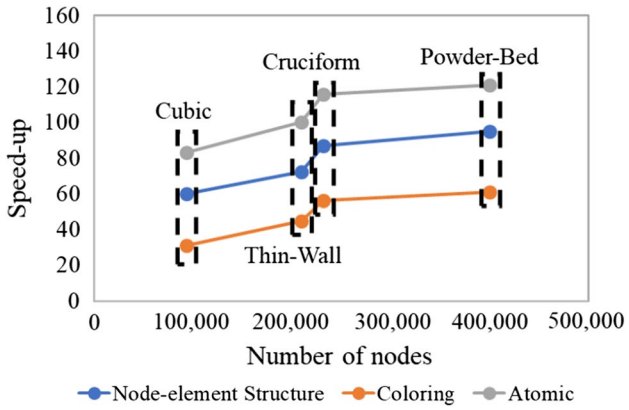


Fig. 10 Correlation between the number of nodes and the achieved speed-up in test samples

registers into local memory, which has a high latency. Constant memory is used for accessing material properties such as density, solidus and liquidus temperatures, specific heat, etc. Since all the threads will load these variables together, the constant memory will broadcast the corresponding values to all the threads at the same time and cause a desirable access pattern. Shared memory is used for calculating the shape function and the Jacobian of each element. This is because these variables are called many times inside the kernels. Having them in the lowest latency memories is essential since keeping them in registers would use too many registers in each block and limit the number of warps that can be executed in each block.

Another major acceleration consideration implemented in this work is to asynchronously launch kernels using CUDA streams to overlap calculations done on the CPU and GPU, overlap data transfer and kernel execution, and concurrently execute GPU kernels. As mentioned before, kernel execution boundaries need to be calculated before execution of each kernel on the CPU. By overlapping these calculations with previous kernel executions both the CPU and GPU can work at the same time to completely hide the time needed for CPU calculation. Overlapping data transfer with kernel execution decreases the time needed for saving the simulation outputs by concurrently copying data from GPU global memory to CPU accessible memory (RAM) while continuing the calculation of the next time step. Finally, concurrent execution of GPU kernels increases the device occupancy by increasing the number of warps scheduled to be run. The asynchronous execution of kernels on different CUDA streams and the overlap between data transfer and device calculations is

demonstrated in Fig. 6, which is the output of the NVIDIA Visual Profiler tool.

As shown in Fig. 6, the data transfer between the GPU and CPU memories is performed concurrently with the initialization of the FEA matrices, which means that this data transfer does not add significant overhead to the simulation time and can be performed as frequently as desired. This technique allows for the transfer of the calculations that are inherently conditional such as computing the cooling rate to the CPU side and achieve further acceleration by avoiding costly conditional computations on the GPU.

5 Acceleration results and verification

To test the acceleration strategies discussed previously, four samples of AM processes are investigated. The samples include three DED builds of cubic, cruciform, and thin-walled geometries, and a powder-bed SLM build. The mesh and geometry of the samples are demonstrated in Fig. 7, where the blue meshes represent the substrate while the red meshes represent the build. The difference between the simulation setup of the SLM process and DED processes is that for SLM, an entire layer of elements is born at the same time, while for DED, the elements are born gradually following the laser focal point.

The simulation parameters of the samples considered for the verification of the proposed algorithms include the number of elements in the range of around 80,000–400,000 elements, the build time in the range of 42–3,007 s, and the stainless steel 316L and Titanium alloy Ti–6Al–4V materials as listed in Table 1.

To determine the effect of assembly strategy on acceleration, the three GPU assembly approaches described in Sect. 4.2 are used to simulate each sample and are compared with an optimized single CPU core implementation of the same calculations. The optimized CPU implementation considers elements as non-deformable and material properties as fixed values. Using these simplifying assumptions, the CPU implementation calculates element and surface Jacobians as well as the element local conduction matrices only once and uses the stored values at each time-step. This implementation is used to enable simulation of the samples in a feasible time frame since the version without simplification is an order of magnitude more computationally expensive. However, all GPU implementations perform these calculations at each time-step which makes them suitable for simulation with deformable elements and temperature-dependent material properties. To have a fair comparison, all calculations in the GPU and CPU implementations are performed based on the explicit FEA formulation presented in 0 using single precision 32-bit floating-point numbers.

The node-element data structure GPU implementation output for the temperature field of the samples during the build is visualized in Fig. 8, in which the range of color bars is set from 300 K to the liquidus temperature of the material; therefore, the red color region represents the melt pool.

The results of the simulation speed-up of the GPU enabled implementations for the assembly strategies compared to the optimized single CPU case are provided in Fig. 9. The results are produced using the NVIDIA GeForce GTX TITAN Black graphics card, which has 2880 CUDA cores, bus support of PCI Express 3.0, 6 GB of global memory, and compute capability of 3.5. The CPU used for this work is an Intel(R) Xeon(R) CPU E5-2687 W with a clock speed of 3.10 GHz.

The major time-consuming operations in this process are calculating the capacitance, conduction flux, and external fluxes with 45.8, 32.8, and 20.1% of the total simulation time, respectively, determined as the average of all test samples. Other operations such as initializations, forward time integration and Dirichlet boundary assignment take on the average less than 2% of the total time.

Although many factors can affect the speed-up of a simulation such as the frequency of output, the distribution of boundary conditions between surfaces, and the birth strategy of the build, these results indicate a correlation between the size of the simulations, which can be represented by the number of elements or nodes, and the speed-up, which is demonstrated in Fig. 10. This is because the more elements and nodes the model has, the more parallel works exist for the GPU and the overall simulation becomes more suitable for the massively parallel architecture of the GPU.

The thin wall and cruciform builds have a similar number of nodes, but there is a significant difference between their speed-ups. As can be seen in Fig. 7, a larger portion of the total nodes of the geometry is associated with the build in the thin-wall simulation with respect to the cruciform

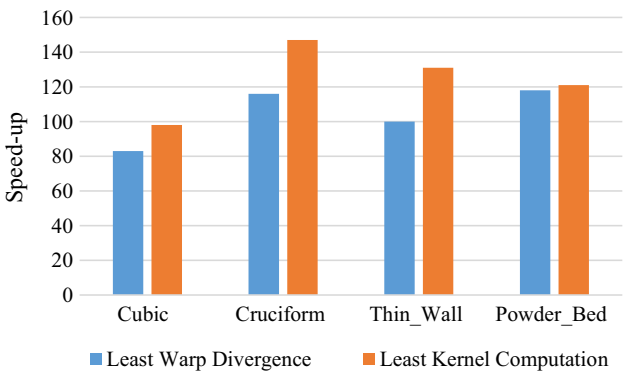


Fig. 11 Acceleration results of the flux calculation strategy for test samples

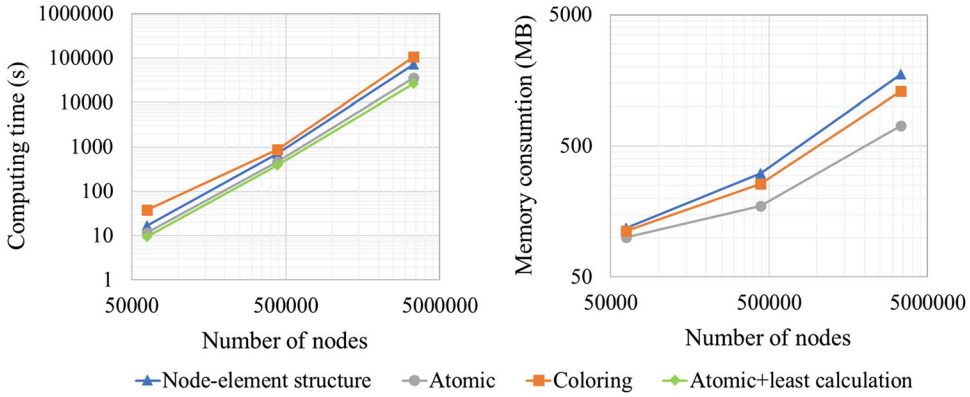


Fig. 12 Computing time and memory consumption of test samples with coarse, medium, and fine meshes with respect to the number of nodes on the log–log scale

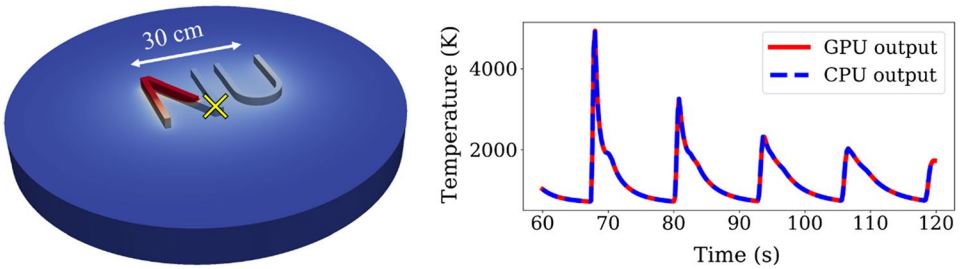


Fig. 13 Validation of the accuracy of the GPU calculations; **a** a demonstration of the test geometry and a screenshot of its thermal profile during the build, where the yellow cross represents the probe point, and **b** the comparison between the GPU and CPU outputs for the thermal hisotry at the probe point

simulation. Considering that the simulation works on active elements and nodes during each time step, the effective size of the simulation for the thin-wall is significantly smaller than for the cruciform build at the beginning time steps, which is one reason for the better performance of the cruciform build.

The assembly strategy using coloring leads to the worst performance between the investigated approaches. This poor performance is because the size of the problem executed on the GPU at each time is dramatically decreased (the decrease is more severe in case of highly unstructured meshes) as it divides the problem domain into multiple smaller subdomains. Furthermore, the explicit flow control requirements to ensure each sub-domain is executed in sequence significantly restricts the amount of parallelization that the GPU can apply to solve the problem. Therefore, the occupancy of the GPU decreases. This is the main bottleneck of this strategy. This bottleneck can be qualitatively seen from the average occupancy of 53.4% of the device for the test samples, while it is over 80% for other strategies. Moreover, operations such as dynamic assignment of boundaries which needs to be repeated for each subdomain and the overhead of initial arrangement of element colors harm the performance of this scheme.

The strategy using the node-element data structure outperforms the coloring approach. However, this strategy has two major computational bottlenecks. First, the size of the matrices for storing capacitance, conduction and external fluxes in global memory is several times larger than the reduced matrices and therefore this strategy leads to far higher requests for global memory access as compared to methods that directly work on reduced matrices. This bottleneck is verified by the NVIDIA profiling tool which indicates that 'load' and 'save' instructions for global memory are the leading bottleneck of this process. Second, this strategy requires separate kernels for reduction operations which slow down the computations in each time step and account for 11% of the execution time on average for the presented test samples.

Table 2 Accuracy of the GPU strategies with respect to the CPU calculations for the NU-shape build

Strategy	Node-element structure and least divergence	Atomic and least divergence	Coloring and least divergence	Atomic and least computation
Mean absolute error	8.538e−6	7.088e−6	9.360e−6	7.096e−6

The strategy using Atomic operations consistently provides the best performance for all the samples. The bottleneck of this strategy is the arithmetic operations which are implicitly halted to control race conditions. However, a race condition does not happen in the assembly procedure for all the nodes since the physical execution of warps can happen

at separate times. Using the Atomic operations allows the GPU to halt only the memory operations with race conditions and avoid costly explicit synchronization. This capability is particularly improved in recent GPUs with Compute Capability of 3 × or higher [48]. The Atomic strategy not only has lower execution time with respect to the other strategies, it also requires the least amount of preprocessing and global memory storage.

The results of the flux calculation strategies are provided in Fig. 11 using the Atomic assembly strategy. The results show that for the DED test samples using the least kernel computation strategy would lead to a significant increase in the performance with respect to the least warp divergence strategy, while the two strategies perform almost identically for the SLM test sample. This dissimilarity can be explained based on the different element generation modes in DED and SLM simulations. In DED processes, the elements are generated gradually by following the laser focal point, while in SLM the whole layer of elements is generated simultaneously. The gradual generation of elements in the DED processes results in a far greater number of external surfaces in the simulations as compared to SLM processes. This result indicates that in the case of powder-bed simulations the cost associated with warp divergence and redundant kernel computations is balanced. However, in the case of DED processes which have more dynamic surfaces the cost of excessive redundant calculations exceeds the warp divergence penalty. Therefore, considering the large number of surface births and deaths in DED processes, it is beneficial to use conditional statements to avoid redundant calculations for inactive surfaces.

To analyze the effect of the number of degrees of freedom on the computing time and memory consumption for the discussed strategies, the cubic sample geometry (Fig. 7a) is tested with three meshes: a coarse (about 60 K nodes), a medium (about 400 K nodes), and a fine mesh (about 3.3 M nodes). The results of this analysis are demonstrated in Fig. 12 on the log–log scale. As it can be seen from Fig. 12 left, the computing time of the four discussed strategies behaves consistently across different numbers of nodes in the numerical model. In terms of memory consumption (Fig. 12 right), Atomic operation consistently leads to the best memory consumption while the node-element structure needs the most memory. Note that due to the GPU memory overhead we observe smaller difference in memory consumption of the strategies for the coarsest mesh. Since the flux calculation strategy between least calculation and least divergence does not affect memory consumption, it is not plotted in Fig. 12 right.

The accuracy of the results is validated by comparing the temperature outputs for the GPU implementations and the CPU one. This validation is demonstrated on a NU-shape build with nearly 200,000 nodes and 150 s of the build time as depicted in Fig. 13a), where the yellow cross represents the probe point. A comparative figure for the output temperature of the probe point calculated on the CPU and the GPU implementation with Atomic operation and the least kernel computation strategies is shown as Fig. 13b), which verifies the accuracy of the GPU calculations. The mean absolute error of different strategies is summarized in Table 2 for the NU-shaped sample, which indicates the correctness of the calculations presented in this work considering that the operations are done on 32 bits floating point numbers with 6 significant decimal digits.

6 Conclusions and Future works

In conclusion, this paper presents methodologies for accelerating the FEA calculations for explicit thermal analysis of metal powder-based AM processes. Three different strategies to avoid race conditions are investigated, namely node-element structure, coloring and atomic. Despite promising results in the literature for assembly using the node-element structure and coloring techniques, our results indicate that atomic operations can lead to the best speed-ups for FEM simulation of AM processes. Considering that AM simulations often start with a fraction of active elements, the coloring strategy does not provide a competitive performance because of its sensitivity on the simulation size. The node-element strategy multiplies the required global memory accesses, which makes memory operations the bottleneck of the process especially for non-uniform mesh structures. Considering that the race condition does not happen for all the elements, using the atomic operations modern GPUs are capable of efficiently halting memory accesses that cause race conditions without explicit costly synchronization. The investigation of two flux calculation strategies (i.e., least warp divergence and least kernel computation) indicates that the cost of excessive computations surpasses the penalty of warp divergence for the FEM simulation of AM processes, especially for DED processes where elements are dynamically born at each time step of the simulation. Furthermore, memory hierarchy and host-device concurrency are used to optimize the use of GPU resources. The implementations are tested on multiple builds which lead to speed-ups of about 100–150 × with respect to an optimized single CPU core implementation for the strategy of assembling by using Atomic operations and flux calculations using the least kernel computation approach along with the proposed optimization.

In the future, more attempts will be dedicated to adding multi-physics formulations into the GPU accelerated FEM model. An important coupled physics with the thermal analysis of the AM process is the thermoelastic behavior of the material that generates deformations and residual stresses.

Additionally, the scalability of the developed FEM package on multi-GPU clusters is another interesting subject that needs further investigation as it is noted in [49–52]. Considering that GPU clusters are becoming increasingly popular, the scalability performance of the model is critical to problems that require large memory or computational power.

Acknowledgements The authors acknowledge the support by the National Institute of Standards and Technology (NIST)—Center for Hierarchical Materials Design (CHiMaD) under Grant No. 70NANB14H012, and the National Science Foundation (NSF)—Cyber-Physical Systems (CPS) under Grant No. CPS/CMMI-1646592. Stephen Lin is supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1324585.

References

1. Yang L, Harrysson O, West H, Cormier D (2012) Compressive properties of Ti–6Al–4V auxetic mesh structures made by electron beam melting. Acta Mater 60(8):3370–3379
2. Guo C, Ge W, Lin F (2015) Dual-material electron beam selective melting: hardware development and validation studies. Engineering 1(1):124–130
3. Wenjun G, Chao G, Feng L (2015) Microstructures of components synthesized via electron beam selective melting using blended pre-alloyed powders of Ti6Al4V and Ti45Al7Nb. Rare Metal Mater Eng 44(11):2623–2627
4. Tan X, Kok Y, Tan YJ, Descoins M, Mangelinck D, Tor SB, Leong KF, Chua CK (2015) Graded microstructure and mechanical properties of additive manufactured Ti–6Al–4V via electron beam melting. Acta Mater 97:1–16
5. Dehoff R, Kirka M, Sames W, Bilheux H, Tremsin A, Lowe L, Babu S (2015) Site specific control of crystallographic grain orientation through electron beam additive manufacturing. Mater Sci Technol 31(8):931–938
6. Gibson I, Rosen DW, Stucker B (2010) Sheet lamination processes. In: Additive manufacturing technologies. Springer, pp 223–252
7. Gu D, Meiners W, Wissenbach K, Poprawe R (2012) Laser additive manufacturing of metallic components: materials, processes and mechanisms. Int Mater Rev 57(3):133–164
8. King W, Anderson A, Ferencz R, Hodge N, Kamath C, Khairallah S, Rubenchik A (2015) Laser powder bed fusion additive manufacturing of metals; physics, computational, and materials challenges. Appl Phys Rev 2(4):041304
9. Parry L, Ashcroft I, Wildman RD (2016) Understanding the effect of laser scan strategy on residual stress in selective laser melting through thermo-mechanical simulation. Addit Manuf 12:1–15
10. Schoinochoritis B, Chantzis D, Salonitis K (2017) Simulation of metallic powder bed additive manufacturing processes with the finite element method: a critical review. Proc Inst Mech Eng Part B: J Eng Manuf 231(1):96–117
11. Khairallah SA, Anderson AT, Rubenchik A, King WE (2016) Laser powder-bed fusion additive manufacturing: physics of complex melt flow and formation mechanisms of pores, spatter, and denudation zones. Acta Mater 108:36–45
12. Rai A, Markl M, Körner C (2016) A coupled cellular automaton-lattice Boltzmann model for grain structure simulation during additive manufacturing. Comput Mater Sci 124:37–48
13. Yan W, Lin S, Kafka OL, Lian Y, Yu C, Liu Z, Yan J, Wolff S, Wu H, Ndip-Agbor E (2018) Data-driven multi-scale multi-physics models to derive process–structure–property relationships for additive manufacturing. Comput Mech 61:1–21
14. Wolff SJ, Lin S, Faierson EJ, Liu WK, Wagner GJ, Cao J (2017) A framework to link localized cooling and properties of directed energy deposition (DED)-processed Ti–6Al–4V. Acta Mater 132:106–117
15. Francois MM, Sun A, King WE, Henson NJ, Tourret D, Bronkhorst CA, Carlson NN, Newman CK, Haut T, Bakosi J (2017) Modeling of additive manufacturing processes for metals: challenges and opportunities. Curr Opin Solid State Materials Sci 21(LA-UR-16-24513)
16. NVIDIA (2016) NVIDIA GPU accelerated applications catalog
17. Tajdari M, Tai BL (2016) Modeling of brittle and ductile materials drilling using smoothed-particle hydrodynamics. In: ASME 2016 11th international manufacturing science and engineering conference, 2016. American Society of Mechanical Engineers
18. Bell N, Hoberock J (2011) Thrust: a productivity-oriented library for CUDA. In: GPU computing gems Jade edition. Elsevier, pp 359–371
19. Bolz J, Farmer I, Grinspun E, Schröoder P (2003) Sparse matrix solvers on the GPU: conjugate gradients and multigrid. In: ACM transactions on graphics (TOG). ACM
20. Nvidia C (2014) Cusparse library. NVIDIA Corporation, Santa Clara
21. Price AD (2013) Multi-GPU Computing with Abaqus: benchmarking and scaling for multiphysics applications in mechatronics
22. Lukarski D (2015) Paralution-library for iterative sparse methods
23. Pichler F, Haase G (2019) Finite element method completely implemented for graphic processor units using parallel algorithm libraries. Int J High Perf Comput Appl 33(1):53–66
24. Cecka C, Lew AJ, Darve E (2011) Assembly of finite element methods on graphics processors. Int J Numer Meth Eng 85(5):640–669
25. Markall G, Slemmer A, Ham D, Kelly P, Cantwell C, Sherwin S (2013) Finite element assembly strategies on multi-core and many-core architectures. Int J Numer Meth Fluids 71(1):80–97
26. Markall GR, Ham DA, Kelly PH (2010) Towards generating optimised finite element solvers for GPUs from high-level specifications. Proc Comput Sci 1(1):1815–1823
27. Dziekonski A, Lamecki A, Mrozowski M (2011) A memory efficient and fast sparse matrix vector product on a GPU. Prog Electromagn Res 116:49–63
28. Dziekonski A, Lamecki A, Mrozowski M (2016) GPU-accelerated finite element method. In: 2016 IEEE MTT-S international conference on numerical electromagnetic and multiphysics modeling and optimization (NEMO). IEEE
29. Dziekonski A, Sypek P, Lamecki A, Mrozowski M (2012) Finite element matrix generation on a GPU. Prog Electromagn Res 128:249–265
30. Saad Y (2003) Iterative methods for sparse linear systems, vol 82. SIAM
31. Knepley MG, Rupp K, Terrel AR (2016) Finite element integration with quadrature on the GPU. arXiv preprint arXiv:1607.04245
32. Knepley MG, Terrel AR (2013) Finite element integration on GPUs. ACM Trans Math Softw (TOMS) 39(2):10
33. Georgescu S, Chow P, Okuda H (2013) GPU acceleration for FEM-based structural analysis. Arch Comput Methods Eng 20(2):111–121
34. Van Belle L, Vansteenkiste G, Boyer JC (2012) Comparisons of numerical modelling of the selective laser melting. In: Key engineering materials. Trans Tech Publ

35. Zaeh MF, Branner G (2010) Investigations on residual stresses and deformations in selective laser melting. Prod Eng Res Dev 4(1):35–45
36. Wang Z, Beese AM (2017) Effect of chemistry on martensitic phase transformation kinetics and resulting properties of additively manufactured stainless steel. Acta Mater 131:410–422
37. Belytschko T, Liu WK, Moran B, Elkhodary K (2013) Nonlinear finite elements for continua and structures. Wiley, Hoboken
38. Fish J, Belytschko T (2007) A first course in finite elements. Wiley, Hoboken
39. Smith J, Xiong W, Cao J, Liu WK (2016) Thermodynamically consistent microstructure prediction of additively manufactured materials. Comput Mech 57(3):359–370
40. Golub GH, Welsch JH (1969) Calculation of Gauss quadrature rules. Math Comput 23(106):221–230
41. Zhu J (2013) The finite element method: its basis and fundamentals. Elsevier, Amsterdam
42. Yan W, Lin S, Kafka OL, Lian Y, Yu C, Liu Z, Yan J, Wolff S, Wu H, Ndip-Agbor EJCM (2018) Data-driven multi-scale multi-physics models to derive process–structure–property relationships for additive manufacturing. Comput Mech 61(5):521–541
43. Mozaffar M, Paul A, Al-Bahrani R, Wolff S, Choudhary A, Agrawal A, Ehmann K, Cao JJMI (2018) Data-driven prediction of the high-dimensional thermal history in directed energy deposition processes via recurrent neural networks. Manuf Lett 18:35–39
44. Cheng J, Grossman M, McKercher T (2014) Professional Cuda C programming. Wiley, Hoboken
45. NVIDIA (2008) NVIDIA CUDA C programming guide, pp. 1–261
46. Lee C-C, Lee D-T (1985) A simple on-line bin-packing algorithm. J ACM (JACM) 32(3):562–572
47. Graham RL (1969) Bounds on multiprocessing timing anomalies. SIAM J Appl Math 17(2):416–429
48. NVIDIA (2018) Features and technical specifications. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capabilities
49. Meng H-T, Nie B-L, Wong S, Macon C, Jin J-MJIA, Magazine P (2014) GPU accelerated finite-element computation for electromagnetic analysis. IEEE Antennas Propag Mag 56(2):39–62
50. Wang H, Zeng Y, Li E, Huang G, Gao G, Li GJCMIAM (2016) "Seen Is Solution" a CAD/CAE integrated parallel reanalysis design system. Comput Methods Appl Mech Eng 299:187–214
51. Zhang R, Wen L, Naboulsi S, Eason T, Vasudevan VK, Qian DJCM (2016) Accelerated multiscale space–time finite element simulation and application to high cycle fatigue life prediction. Comput Mech 58(2):329–349
52. Yamaguchi T, Fujita K, Ichimura T, Hori T, Hori M, Wijerathne LJPCS (2017) Fast finite element analysis method using multiple gpus for crustal deformation and its application to stochastic inversion analysis with geometry uncertainty. Proc Comput Sci 108:765–775
53. Bennett JL, Wolff SJ, Hyatt G, Ehmann K, Cao J (2017) Thermal effect on clad dimension for laser deposited Inconel 718. J Manuf Process 28:550–557
54. Commons W (2015) File: selective laser melting system schematic.jpg—Wikimedia Commons{,} the free media repository. https://commons.wikimedia.org/w/index.php?title=File:Selective_laser_melting_system_schematic.jpg&oldid=154088078. Accessed 15 Oct 2018