Alternative Implementation of Hotspot3D using CUDA Matrix Libraries

Aritra Bhakat Ludwig Kristoffersson Saga Palmér

January 5, 2024

The repo containing our implementation can be found on GitHub: hotspot3d-matrix

Group Member Contributions

Aritra Bhakat

Ludwig Kristoffersson

Saga Palmér

Introduction

Hotspot3d tries to solve the heat equation, given by the following PDE:

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T(\mathbf{x}) + \beta P(\mathbf{x}) + \alpha \left[T_{\text{amb}} - T(\mathbf{x}) \right] \tag{1}$$

$$= \alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right) + \beta P(\mathbf{x}) + \alpha \left[T_{\text{amb}} - T(\mathbf{x}) \right] \tag{2}$$

where T is the temperature field, P is the power field, T_{amb} is the ambient temperature, and α, β are arbitrary constants which have to do with chip properties.

The PDE is discretised using forward difference for the time step and second order central differences for the Laplace operator:

$$
\begin{aligned}
T^{t+1}_{ijk} = & && (3)\\
T^t_{ijk} + \Big[c_w T^t_{i-1,jk} - (c_w + c_e)T^t_{ijk} + c_e T^t_{i+1,jk}\Big] & && (4)\\
+ \Big[c_t T^t_{i,j-1,k} - (c_t + c_s)T^t_{ijk} + c_s T^t_{i,j+1,k}\Big] & && (5)\\
+ \Big[c_b T^t_{ij,k-1} - (c_b + c_t)T^t_{ijk} + c_t T^t_{ij,k+1}\Big] & && (6)\\
+\text{sdc} \cdot P_{ijk} & && (7)\\
+c_t[T^t_{\text{amb}} - T^t_{ijk}] & && (8)\\
= c_c T^t_{ijk} & && (9)\\
+c_w T^t_{i-1,jk} + c_e T^t_{i+1,jk} & && (10)\\
+c_t T^t_{i,j-1,k} + c_s T^t_{i,j+1,k} & && (11)\\
+c_b T^t_{ij,k-1} + c_t T^t_{ij,k+1} & && (12)\\
+\text{sdc} \cdot P_{ijk} + c_t T^t_{\text{amb}} & && (13)
\end{aligned}
$$

Here, the constants c_x are combinations of the constant α_x, along with the timestep dt, size of the chip in the given dimension and amount rows/cols/layers for the given dimension. The letters w, e, n, s, b, t stand for the compass dimensions and bottom and top, each pair of irections corresponding the xyz axes. The values are symmetrical, meaning $c_w = c_e \ldots$ and so on. To simplify the computation, the T^t_{ijk} terms are gathered into the constant $c_c = 1 - (2 \cdot c_w + 2 \cdot c_n + 3 \cdot c_t)$. The constant sdc equals dt/capacitance. The temperature T and the power P both have dimensions $n_x \times n_y \times n_z$.

Methodology

We observed that the discretised PDE corresponds well to sparse matrix multiplication. Our approach is to use a sparse matrix for each dimension to calculate the second order central difference part of the discretised equation. The rest of the equation can be calculated as trivial (vector) addition.

Our data has the $n_x \times n_y \times n_z$. Our differentiation matrices are sparse square matrices with dimensions n_x, n_y and n_z respectively:

$$
\partial X = \begin{bmatrix}
c_w + c_c & c_e & & & & & 0 \\
c_w & c_c & c_e & & & & \\
 & c_w & c_c & c_e & & & \\
 & & & \cdots & & & \\
 & & & & c_w & c_c & c_e \\
0 & & & & & c_w & c_c + c_e
\end{bmatrix} \tag{14}
$$

$$\partial Y = \begin{bmatrix} c_n & c_s & & & & & 0 \\ c_n & 0 & c_s & & & & \\ & c_n & 0 & c_s & & & \\ & & & \cdots & & & \\ & & & & c_n & 0 & c_s \\ 0 & & & & & c_n & c_s \end{bmatrix} \tag{15}$$

$$\partial Y = \begin{bmatrix} c_b & c_t & & & & & 0 \\ c_b & 0 & c_t & & & & \\ & c_b & 0 & c_t & & & \\ & & & \cdots & & & \\ & & & & c_b & 0 & c_t \\ 0 & & & & & c_b & c_t \end{bmatrix} \tag{16}$$

Note that ∂Y and ∂Z are missing c_c in the diagonal as the term is already accounted for in ∂X. These matrices operate on the 3D temperature data.

Since we can only work with matrices and not directly with 3D tensors within the cuSPARSE library, we need to configure the matrix layouts for the different computations to give us the correct result.

We apply ∂X and ∂Y on n_z batches of $n_x \times n_y$ matrices.

$$T_{XY} = \begin{bmatrix} T_{111} & \dots & T_{1,n_x,1} \\ & \dots & \\ T_{n_y,11} & \dots & T_{n_y,n_x,1} \end{bmatrix} \quad \dots \quad \begin{bmatrix} T_{11,n_z} & \dots & T_{1,n_x,n_z} \\ & \dots & \\ T_{n_y,1,n_z} & \dots & T_{n_y,n_x,n_z} \end{bmatrix} \tag{17}$$

We call this layout T_{XY}.

When we apply ∂X, we need to take the transpose of T_{XY} so that the x-dimension is lined up with the column. We then need to transpose the result so the x dimension is lined up with the row again. We perform the following calculation:

$$(\partial X \cdot T_{XY}^{\intercal})^{\intercal} = T_{XY} \cdot \partial X^{\intercal}$$

For ∂Y the calculation is straightforward:

$$\partial Y \cdot T_{XY}$$

For ∂Z we need to change the layout of the T. We flatten the data into one big $n_z \times (n_x \cdot n_y)$ matrix, so that values adjacent on the z-axis are lined up in one column:

$$\begin{bmatrix} T_{111} & \dots & T_{n_y,n_x,1} \\ T_{112} & \dots & T_{n_y,n_x,2} \\ & \dots & \\ T_{11,n_z} & \dots & T_{n_y,n_x,n_z} \end{bmatrix} \tag{18}$$

We call this layout T_Z. The calculation we perform now is also straightforward:

$$\partial Z \cdot T_Z$$

Putting this together, we can rewrite the discretised equation as:

$$T^{t+1} = T^t_{XY} \cdot \partial X^\intercal + \partial Y \cdot T^t_{XY} + \partial Z \cdot T^t_Z + \mathrm{sdc} \cdot P + c_t T_{\mathrm{amb}} I$$

Experimental Setup

Results

Discussion

References