Sorbonne Université

Calcul haute performance : programmation et algorithmique avancées

Projects: Merge Sort, Monte Carlo and PDE simulations

The presentation of the results on December 14th

The source code must be readable, sufficiently commented.
The source code and slides must be sent before December 13th
Presentation duration = 8 to 10 minutes for results + 4 to 6 minutes for code + 5 minutes of questions

1 Merge large and batch merge small

This subject starts with the same algorithm of merge path, presented in [**?**], implemented on only one block. The students are then asked either to translate it into the merge of large arrays or the batch merge of small ones.

We start with the merge path algorithm. Let A and B be two ordered arrays (increasing order), we want to merge them in an M sorted array. The merge of A and B is based on a path that starts at the top-left corner of the $|A| \times |B|$ grid and arrives at the down-right corner. The Sequential Merge Path is given by Algorithm **??** and an example is provided in Figure **??**.

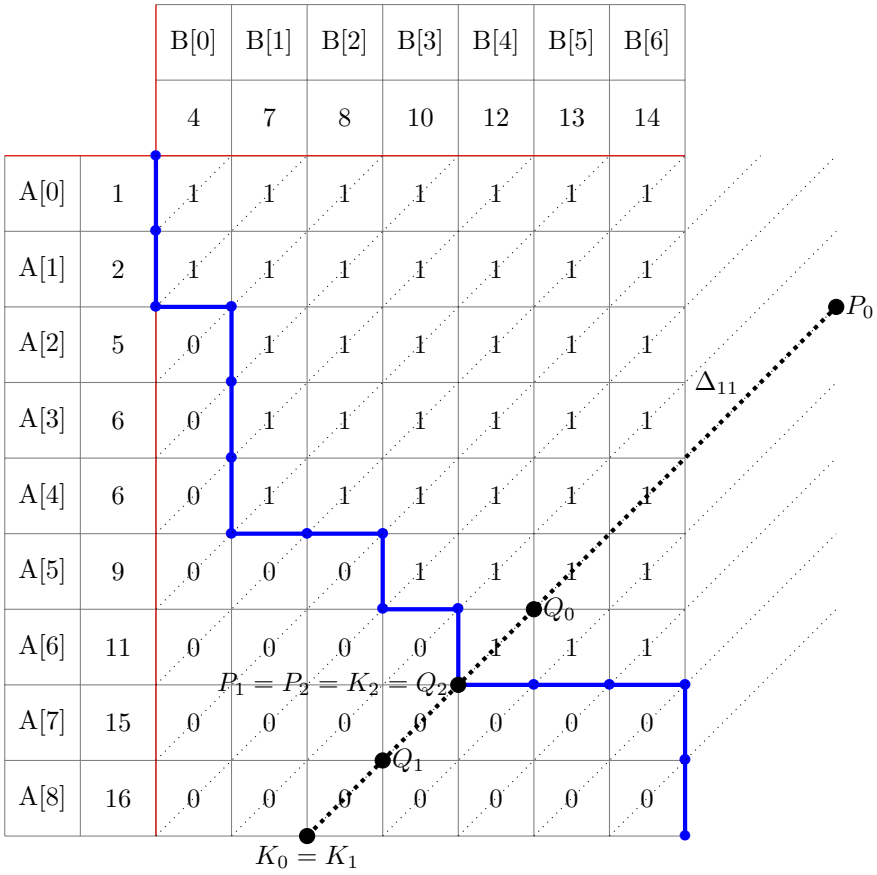

Figure 1: An example of Merge Path procedure

Algorithm 1 Sequential Merge Path

Require: A and B are two sorted arrays
Ensure: M is the merged array of A and B with $|M| = |A| + |B|$
  procedure MERGEPATH (A, B, M)
    $j = 0$ and $i = 0$
    while $i + j < |M|$ **do**
      if $i \geq |A|$ **then**
        M[i+j]=B[j]
        $j = j + 1$ ▷ The path goes right
      else if $j \geq |B|$ or $A[i] < B[j]$ **then**
        M[i+j]=A[i] ▷ The path goes down
        $i = i + 1$
      else
        M[i+j]=B[j]
        $j = j + 1$ ▷ The path goes right
      end if
    end while
  end procedure

Algorithm 2 Merge Path (Indexes of n threads are 0 to $n - 1$)

Require: A and B are two sorted arrays
Ensure: M is the merged array of A and B with $|M| = |A| + |B|$
  for each thread **in parallel do**
    i=index of the thread
    if $i > |A|$ **then**
      $K = (i - |A|, |A|)$ ▷ Low point of diagonal
      $P = (|A|, i - |A|)$ ▷ High point of diagonal
    else
      $K = (0, i)$
      $P = (i, 0)$
    end if
    while True **do**
      $offset = abs(K_y - P_y)/2$
      $Q = (K_x + offset, K_y - offset)$
      if $Q_y \geq 0$ and $Q_x \leq B$ and
        $(Q_y = |A|$ or $Q_x = 0$ or $A[Q_y] > B[Q_x - 1])$ **then**
        if $Q_x = |B|$ or $Q_y = 0$ or $A[Q_y - 1] \leq B[Q_x]$ **then**
          if $Q_y < |A|$ and $(Q_x = |B|$ or $A[Q_y] \leq B[Q_x])$ **then**
            $M[i] = A[Q_y]$ ▷ Merge in M
          else
            $M[i] = B[Q_x]$
          end if
          Break
        else
          $K = (Q_x + 1, Q_y - 1)$
        end if
      else
        $P = (Q_x - 1, Q_y + 1)$
      end if
    end while
  end for

Each point of the grid has a coordinate $(i, j) \in [\![0, |A|]\!] \times [\![0, |B|]\!]$. The merge path starts from the

point $(i,j) = (0,0)$ on the left top corner of the grid. If $A[i] < B[j]$ the path goes down else it goes right. The array $[\![0, |A|-1]\!] \times [\![0, |B|-1]\!]$ of boolean values $A[i] < B[j]$ is not important in the algorithm. However, it shows clearly that the merge path is a frontier between ones and zeros.

To parallelize the algorithm, the grid has to be extended to the maximum size equal to $\max(|A|,|B|) \times \max(|A|,|B|)$. We denote K_0 and P_0 respectively the low point and the high point of the ascending diagonals Δ_k. On GPU, each thread $k \in [\![0, |A|+|B|-1]\!]$ is responsible of one diagonal. It finds the intersection of the merge path and the diagonal Δ_k with a binary search described in Algorithm **??**.

1. For $|A|+|B| \leq 1024$, write a kernel `mergeSmall_k` that merges A and B using only one block of threads.

1.1 Merge large

As mentioned in [**?**], merge path algorithm is divided into 2 stages: partitioning stage and merging stage. The partitioning stage is important to propose an algorithm that involves various blocks.

2. For any size $|A|+|B| = d$ sufficiently smaller than the global memory, write a solution that merges A and B using various blocks.

3. Study the execution time with respect to d.

1.2 Batch merge small and batch sort small

In this part, we assume that we have a large number $N(\geq 1e3)$ of arrays $\{A_i\}_{1\leq i\leq N}$ and $\{B_i\}_{1\leq i\leq N}$ with $|A_i|+|B_i| = d \leq 1024$ for each i. Using some changes on `mergeSmall_k`, we would like to write `mergeSmallBatch_k` that merges two by two, for each i, A_i and B_i.

Given a fixed common size $d \leq 1024$, `mergeSmallBatch_k` is launched using the syntax

```
mergeSmallBatch_k<<<numBlocks, threadsPerBlock>>>(...);
```

with `threadsPerBlock` is multiple of d but smaller than 1024 and `numBlocks` is an arbitrary sufficiently big number.

Try to figure out how the indices
```
int tidx = threadIdx.x%d;
int Qt = (threadIdx.x-tidx)/d;
int gbx = Qt + blockIdx.x*(blockDim.x/d);
```
are important in the definition of `mergeSmallBatch_k`.

5. Write the kernel `mergeSmallBatch_k` that batch merges two by two $\{A_i\}_{1\leq i\leq N}$ and $\{B_i\}_{1\leq i\leq N}$. Study the execution time with respect to d.

6. Write the kernel `sortSmallBatch_k` that sorts various arrays $\{M_i\}_{1\leq i\leq N}$ of size $d \leq 1024$. Study the execution time with respect to d.

2 Monte Carlo and nested Monte Carlo for bullet options

In mathematical finance, we usually use Black & Scholes (B&S) model for assets dynamic on a time interval $[0,T]$. For time increments s and t with $0 \leq s < t \leq T$, this model can be simulated using the following time induction

$$S_t = S_s \exp\left((r-\sigma^2/2)(t-s) + \sigma\sqrt{t-s}G\right) \text{ and } S_0 = x_0 \text{ where}$$

- σ is the volatility assumed here equal to 0.2,

- r is the risk-free rate assumed here equal to 0.1,
- x_0 is the initial spot price of S at time 0, assumed here equal to 100
- G is independent from S_s and has a standard Normal distribution $\mathcal{N}(0,1)$.

Thus, simulating a Gaussian random variable G, using for example Box-Muller method with two Uniformly distributed random variables on $[0,1]$, we can simulate trajectoies of S for any time schedule $T_0 = 0 < T_1 < ... < T_M = T = T_{M+1}$.

Assuming B&S model, in the following we are going first to use Monte Carlo (MC) simulation in order to approximate the price of a bullet option at time $t = 0$. Then using Nested Monte Carlo (NMC) simulation, we would like to approximate the price of a bullet option for multiple possible increments of time and space. Both MC and NMC simulations require the use of cuRAND for random number generation.

The price of a bullet option $F(t,x,j) = e^{-r(T-t)}E(X\big|S_t = x, I_t = j)$, $X = (S_T - K)_+ 1_{\{I_T \in [P_1,P_2]\}}$ with $I_t = \sum_{T_i \leq t} 1_{\{S_{T_i} < B\}}$ and

- K is the contract's strike assumed here equal to $x_0 = 100$,
- T is the contract's maturity assumed here equal to 1
- barrier B should be bigger than S I_T times $\in \{P_1, ..., P_2\} \subset \{0, ..., M\}$ where P_1 and P_2 are two integers.

First, we want to use MC to approximate $F(0, x_0, 0) = e^{-rT}E(X)$ with $\{X_i\}_{i \leq n}$ being independent random variables that have the same distribution as X. We recall that MC simulation is based on the strong law of large numbers that announces that

$$P\left(\lim_{n \to +\infty} \frac{X_1 + X_2 + ... + X_n}{n} = E(X)\right) = 1$$

and the central limit theorem that gives the speed of convergence

$$\frac{\sqrt{n}}{\sigma}\epsilon_n \to G \sim \mathcal{N}(0,1),\ \sigma \text{ is the standard deviation of } X$$

where $\epsilon_n = E(X) - \dfrac{X_1 + X_2 + ... + X_n}{n}$. Thus, for large n, there is a 95% chance of having: $|\epsilon_n| \leq 1.96\dfrac{\sigma}{\sqrt{n}}$.

1. When $P_1 = P_2 = B = 0$, convince yourself that $E(X) = E((S_T - K)_+)$. In this case, use MC to approximate $F(0, x_0, 0)$ and check that you get a value that is close enough to

$$N(d_1)x_0 - N(d_2)Ke^{-rT} \text{ with } N(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-y^2/2}dy,\ d_1 = \frac{\ln(x_0/K) + (r + \sigma^2/2)(T)}{\sigma\sqrt{T}}, \quad (1)$$

   and $d_2 = d_1 - \sigma\sqrt{T}$. N is `NP` function already given in this course.

In the following, we assume that $B = 120$, $M = 100$, $P1 = 10$, $P2 = 50$ and $T_i = iT/M$ for $i = 0, ..., M$.

2. Write the Monte Carlo simulation code that approximates $F(0, x_0, 0)$.
3. First simulate trajectories of $(S_t, I_t)_{t=0,T_1,...,T_M=T}$. On the top of these outer trajectories and starting at $(k, x, j) \in \{0, 1, ..., M\} \times \mathbb{R}_+ \times \{0, 1, ..., \min(k, P_2)\}$, write NMC code that allows to simulate realizations of $F(T_k, x, j)$.

3 PDE simulation of bullet options

The students have to implement and compare Thomas algorithm to PCR for tridiagonal systems. Then, they have to simulate a PDE of a bullet option using Crank-Nicolson scheme based on either Thomas or PCR.

We refer to [**?**] for a fair description of PCR. Regarding Thomas algorithm, as described in [**?**], it allows to solve tridiagonal systems:

$$\begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & 0 & \\ & a_3 & b_3 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \ddots & \ddots & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{2}$$

using a forward phase

$$c'_1 = \frac{c_1}{b_1},\ y'_1 = \frac{y_1}{b_1},\ c'_i = \frac{c_i}{b_i - a_i c'_{i-1}},\ y'_i = \frac{y_i - a_i y'_{i-1}}{b_i - a_i c'_{i-1}} \text{ when } i = 2, ..., n \tag{3}$$

then a backward one

$$z_n = y'_n,\ z_i = y'_i - c'_i z_{i+1} \text{ when } i = n-1, ..., 1. \tag{4}$$

1. Using Thomas method, write a kernel that solves various tridiagonal systems ($d \leq 1024$) at the same time, one system per block. Do the same thing for PCR then compare both methods.

Let $u(t, x, j) = e^{r(T-t)} F(t, e^x, j)$ where F is the price of a bullet option as in Section **??**. One can then show that, on any interval $t \in [T_{M-k-1}, T_{M-k})$, $k = M-1, ..., 0$, $u(t, x, j)$ is the solution of the PDE

$$\frac{1}{2}\sigma^2 \frac{\partial^2 u}{\partial x^2}(t, x, j) + \mu \frac{\partial u}{\partial x}(t, x, j) = -\frac{\partial u}{\partial t}(t, x, j)$$

$$\text{with: } \mu = r - \frac{\sigma^2}{2}$$

. The final and boundary conditions are:

$$\begin{aligned} \bullet\ u(T, x, j) &= \max(e^x - K, 0) \mathbb{1}_{\{j \in [P_1, P_2]\}} \text{ for any (x, j)} \\ \bullet\ u(t, \log[K/3], j) &= \texttt{pmin} = 0 \\ \bullet\ u(t, \log[3K], j) &= \texttt{pmax} = 2K \end{aligned}$$

In addition, we have discontinuities at time T_{M-k} as

$$\lim_{t \overset{<}{\longrightarrow} T_{M-k}} u_t(x, j) = \begin{cases} u_{T_{M-k}}(S_{T_{M-k}}, P_2) \mathbb{1}_{\{S_{T_{M-k}} \geq B\}} & \text{if } j = P_2 \\ u_{T_{M-k}}(S_{T_{M-k}}, P_k^1) \mathbb{1}_{\{S_{T_{M-k}} < B\}} & \text{if } j = P_k^1 - 1 \\ \begin{bmatrix} u_{T_{M-k}}(S_{T_{M-k}}, j) \mathbb{1}_{\{S_{T_{M-k}} \geq B\}} \\ + u_{T_{M-k}}(S_{T_{M-k}}, j+1) \mathbb{1}_{\{S_{T_{M-k}} < B\}} \end{bmatrix} & \text{if } j \in [P_k^1, P_2 - 1] \end{cases} \tag{5}$$

with $P_k^1 = max(P_1 - k, 0)$.

From now on we use notations $u_t(x, j) = u(t, x, j)$ and $u_{k,i} = u(t_k, x_i, j)$. Following Crank Nicolson scheme, we get

$$q_u u_{k,i+1} + q_m u_{k,i} + q_d u_{k,i-1} = p_u u_{k+1,i+1} + p_m u_{k+1,i} + p_d u_{k+1,i-1}$$

$$q_u = -\frac{\sigma^2 \Delta t}{4\Delta x^2} - \frac{\mu \Delta t}{4\Delta x}, \quad q_m = 1 + \frac{\sigma^2 \Delta t}{2\Delta x^2}, \quad q_d = -\frac{\sigma^2 \Delta t}{4\Delta x^2} + \frac{\mu \Delta t}{4\Delta x}$$
$$p_u = \frac{\sigma^2 \Delta t}{4\Delta x^2} + \frac{\mu \Delta t}{4\Delta x}, \quad p_m = 1 - \frac{\sigma^2 \Delta t}{2\Delta x^2}, \quad p_d = \frac{\sigma^2 \Delta t}{4\Delta x^2} - \frac{\mu \Delta t}{4\Delta x}$$

The figure below shows an example of how PDE's backward resolution algorithm (with M = 10, $P_1 = 3, P_2 = 8$) is deployed with time on the x-axis and the set of values of I_t in the ordinate.

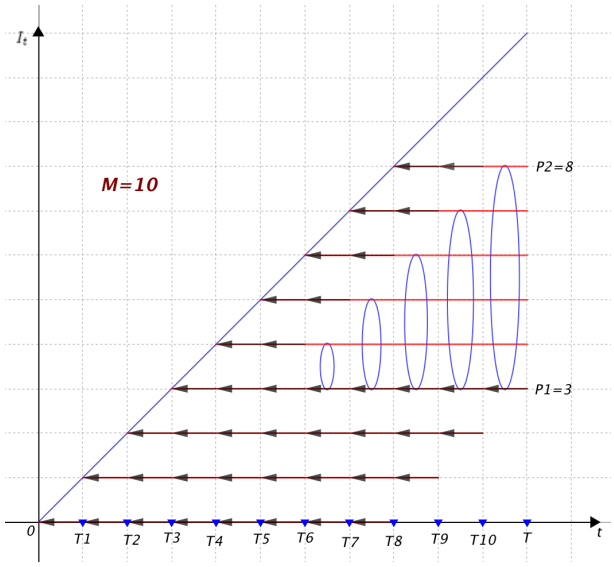

Figure 2: Backward induction scheme

2. Write the code that solves the PDE on $[T_{M-1}, T)$.

3. Write the code that solves all the PDE.

References

[1] O. Green, R. McColl and D. A. Bader GPU Merge Path - A GPU Merging Algorithm. *26th ACM International Conference on Supercomputing (ICS)*, San Servolo Island, Venice, Italy, June 25-29, 2012.

[2] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (2002): *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press.

[3] Y. Zhang, J. Cohen and J. D. Owens (2010): Fast Tridiagonal Solvers on the GPU. *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 127–136.