GPU accelerated RNA folding algorithm

Guillaume Rizk, Sanjay Rajopadhye, Dominique Lavenier

Guillaume.Rizk@irisa.fr

Context

- RNA secondary structure prediction is computationally challenging.
- Used in many bioinformatics pipelines.
- Run times up to several weeks.
- GPU provides cheap performance.

Outline

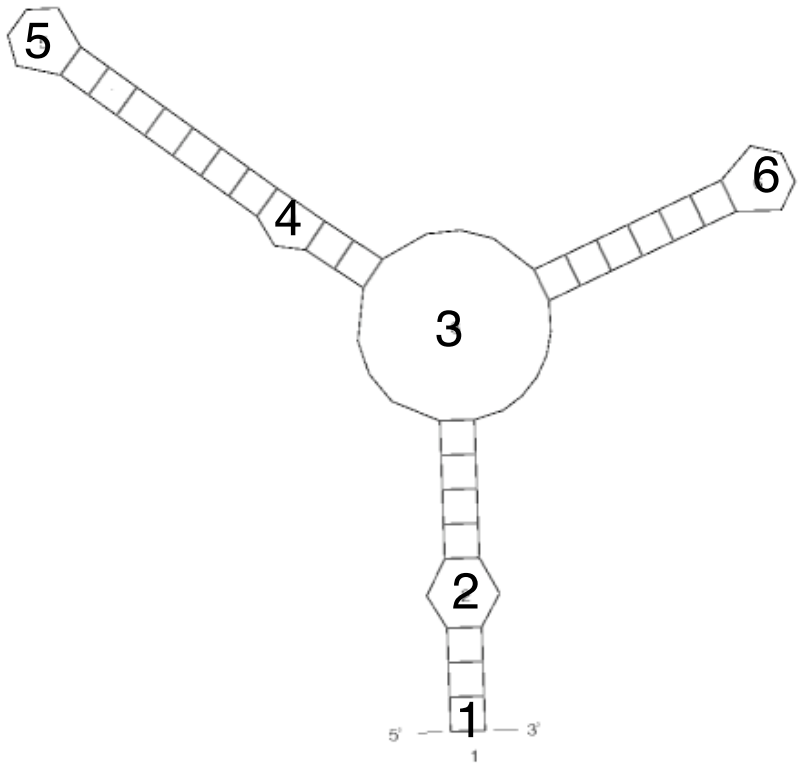
Features

- CUDA : Compute Unified Device Architecture.
- C for CUDA : C language with extensions.
- Massively parallel : 240 processors, thousands of threads.
- Data parallelism.
- Typical speedup : $\times 10$, up to $\times 100$.

Specificities

- Small on chip memory, but latency-hiding mechanism for global accesses.
- Requires thousands of threads.
- Blocks of threads, with shared memory and synchronization.
- High memory bandwidth, but requires special access pattern.
- Relatively slow CPU $\leftrightarrow$ GPU transfers.

RNA Secondary Structure




```
AAAAAAGGGAAAAGAACAAAGGAGACUCUUCUCCUUUUUCAAAGGAAGAGGAGACUCUUUCAAAAAUCCCUCUUUU
((((.(((((...(((.(((((((((....)))))))))))...((((((....))))))).....)))))).)))) (-24.5)
```

Nucleotide pairs allowed : (A-U),(C-G),(G-U)

One structure example among many possibles.

Mfold algorithm

Computes the most stable structure in $\mathcal{O}(n^3)$.

Zuker, M., Stiegler, P. : Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Res 9(1) (1981) 133-148.

Implemented in *hybrid-ss-min* and *RNAfold* functions of Unafold and Vienna RNA packages.

Summarized principle.

- Structure energy = Sum of substructures (loops) energies.
- Thermodynamic model to compute loop energies.
- Explores all possible structures, outputs the one with minimum energy.
- Optimization problem performed via dynamic programming.

Dynamic Programming

3 Tables representing minimum energy of subsequence i, j :

- $Q'_{i,j}$: i and j are paired.
- $Q_{i,j}$: with at least one pair, inside a multiloop.
- $QM_{i,j}$: with at least two pairs , inside a multiloop.

Recursive formulas

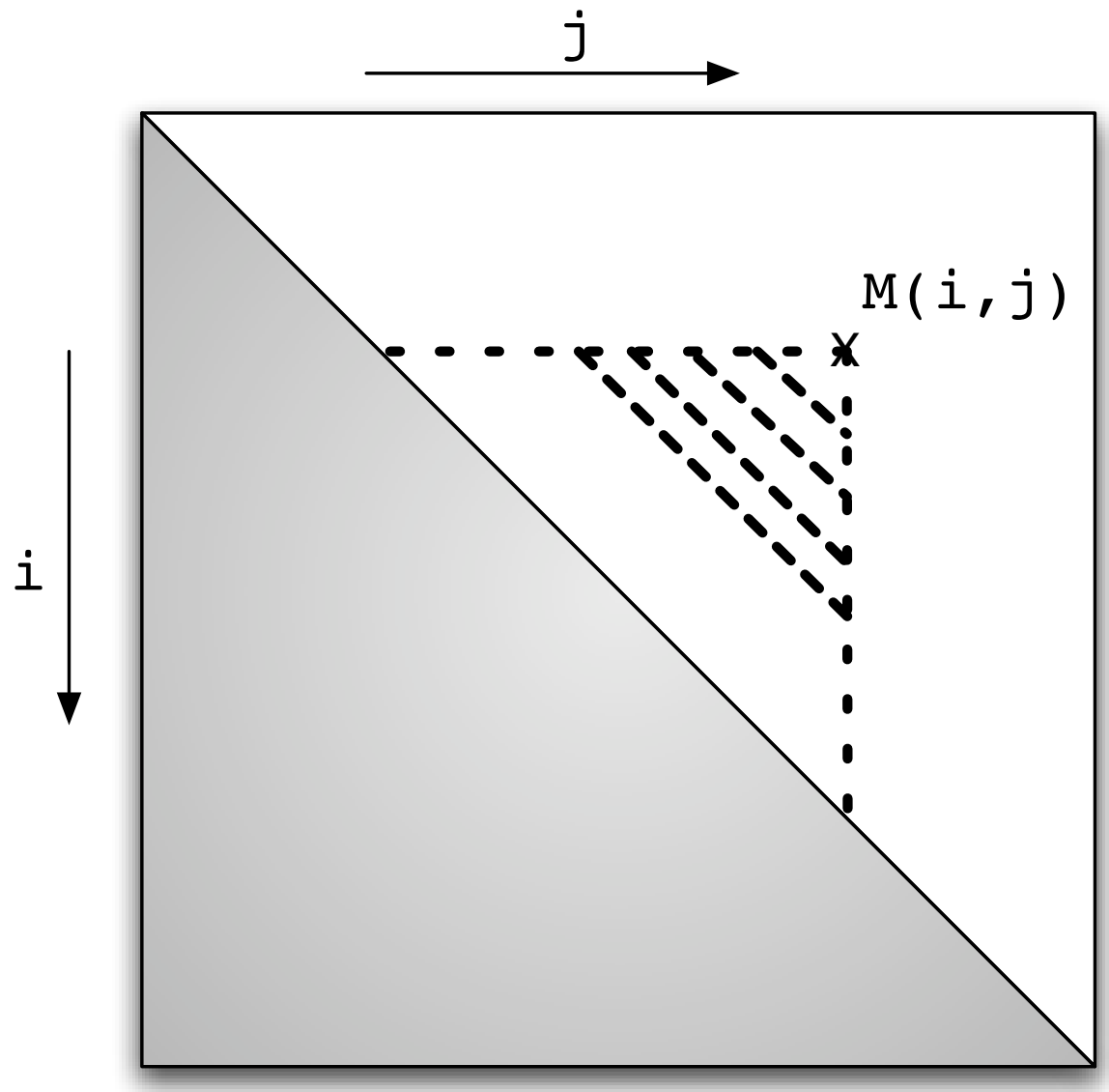
$$Q'_{i,j} = \begin{cases} \min \begin{cases} Eh(i,j) \\ Es(i,j) + Q'_{i+1,j-1} \\ \min\limits_{k,l \in]i;j[^2} Ei(i,j,k,l) + Q'_{k,l} \\ QM_{i+1,j-1} \end{cases} & i \cdot j \text{ allowed} \\ \infty & i \cdot j \text{ not allowed} \end{cases} \quad (1)$$

Recursive formulas

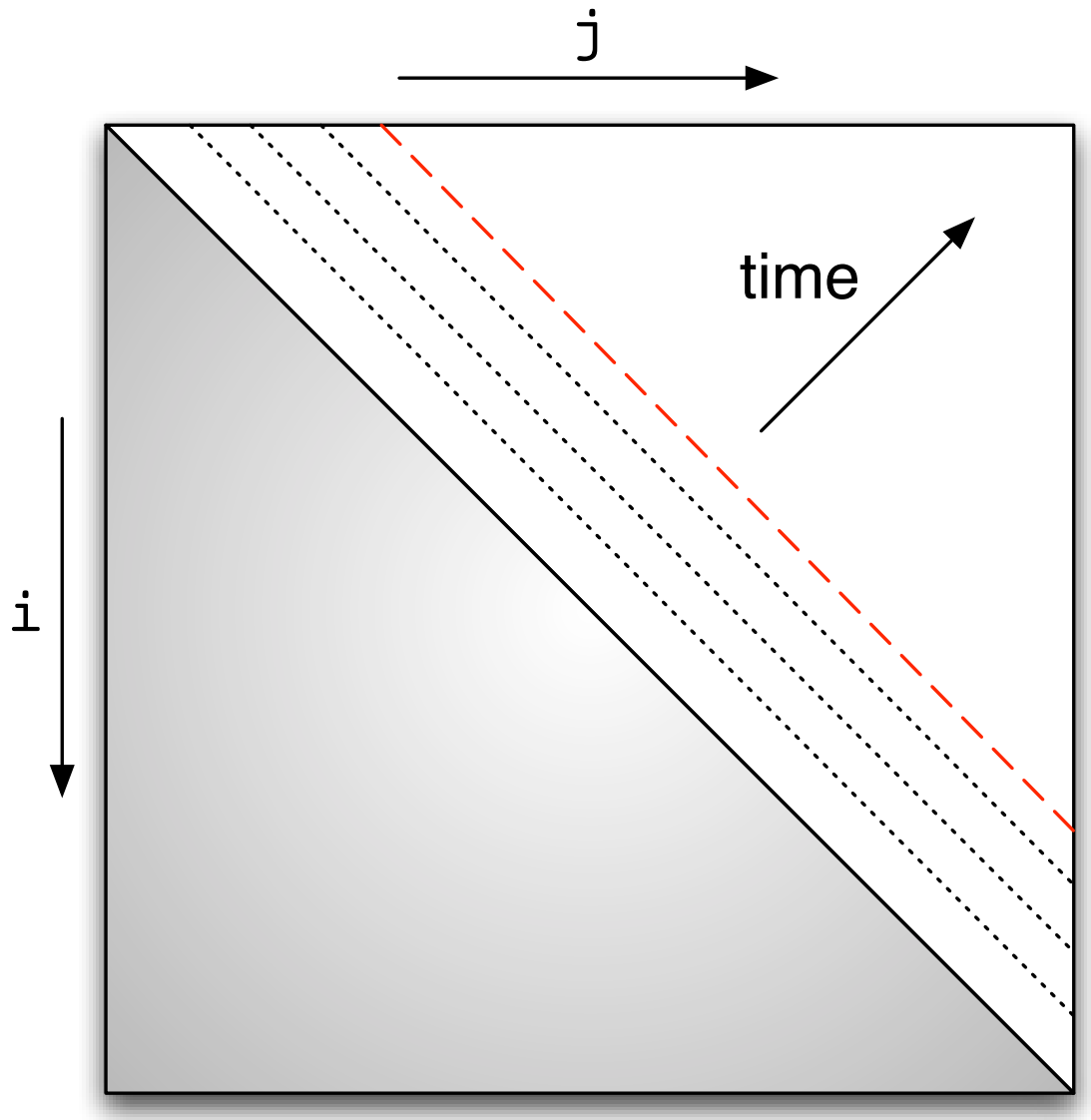
$$QM_{i,j} = \min_{i<k<j} (Q_{i,k} + Q_{k+1,j}) \quad (2)$$

$$Q_{i,j} = \min \begin{cases} QM_{i,j} \\ \min(Q_{i+1,j}, Q_{i,j-1}) \\ Q'_{i,j} \end{cases} \quad (3)$$

Dependency pattern



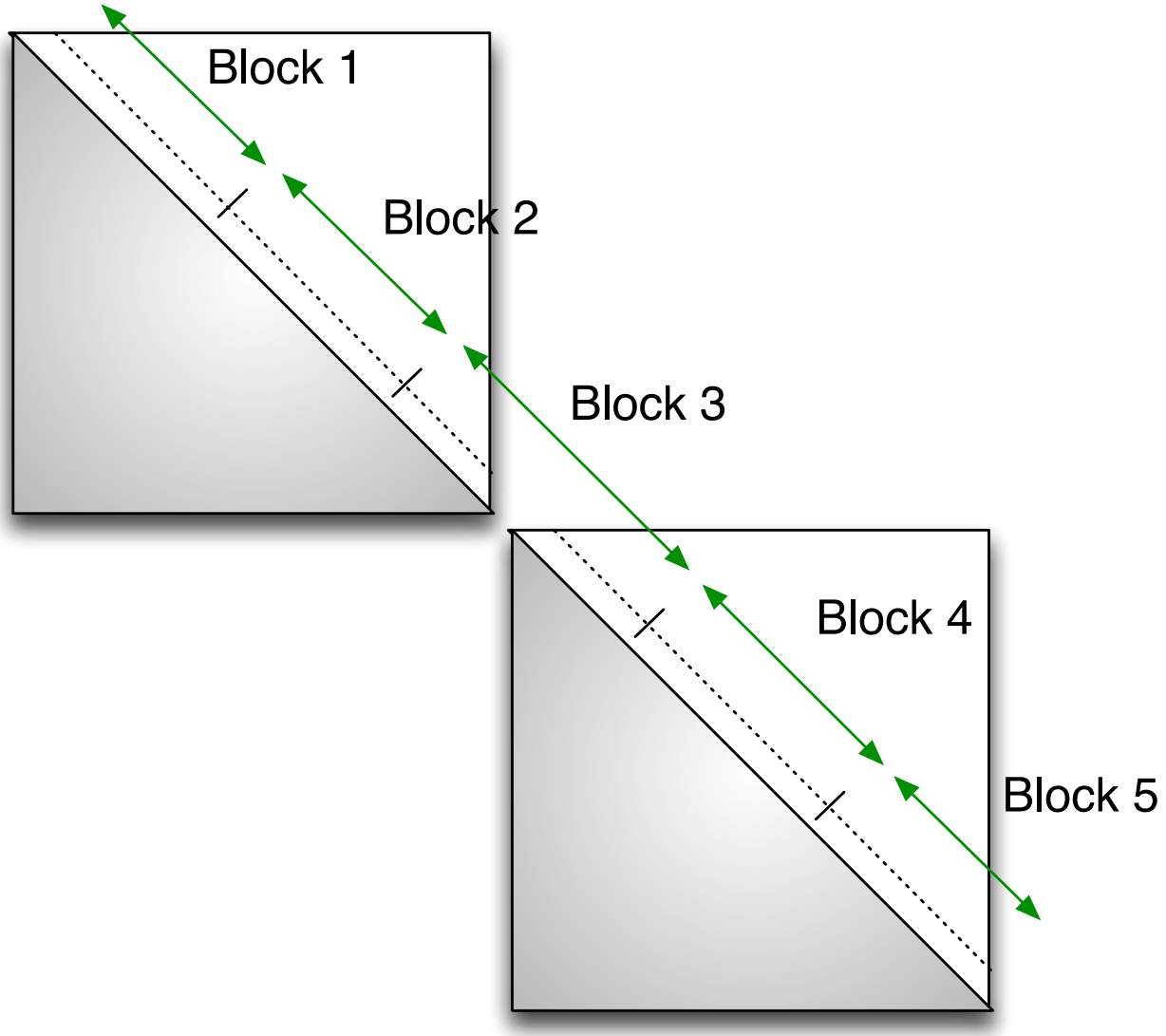
Parallelization scheme



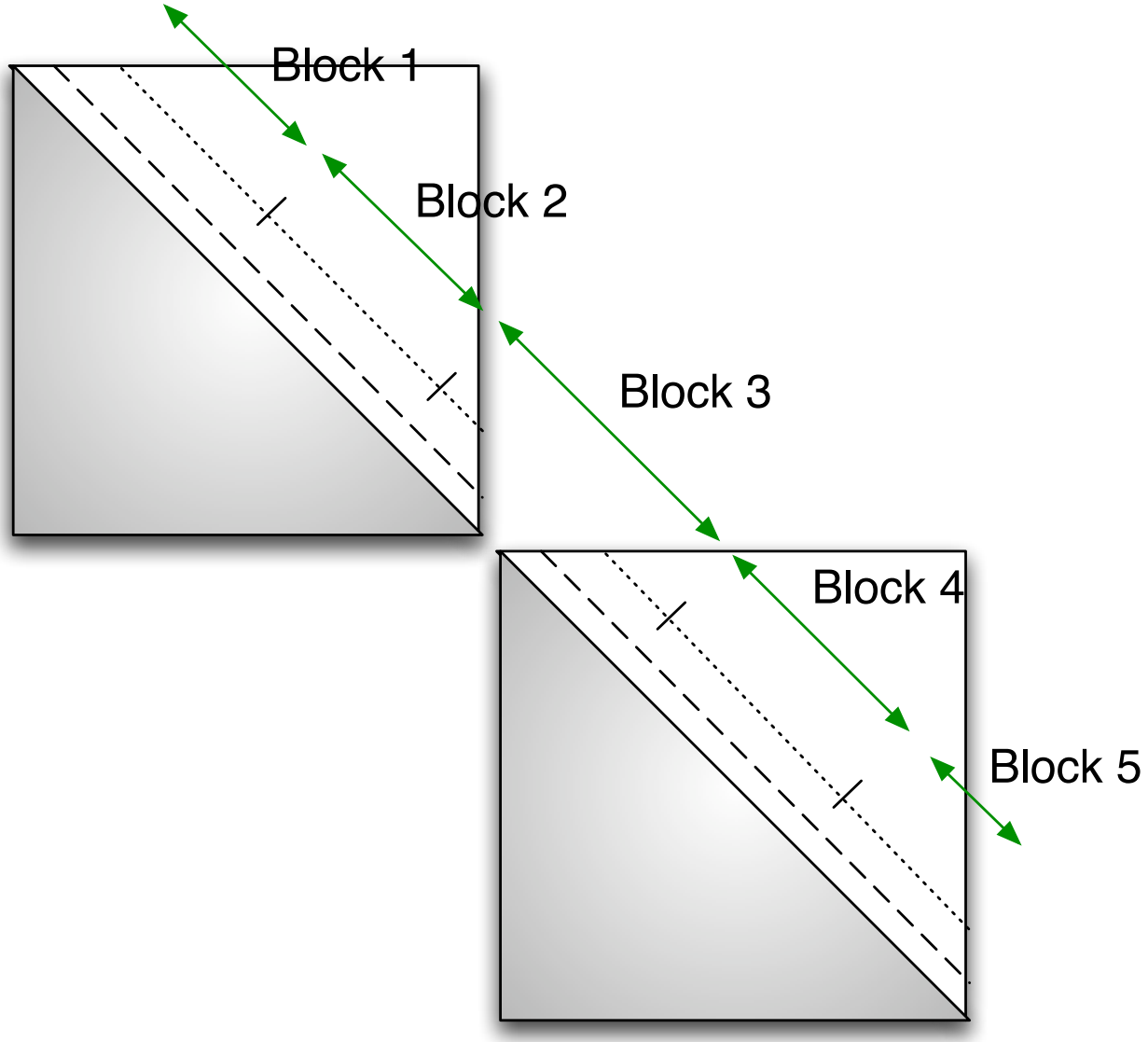
Parallelization Levels

- Coarse grained : across several sequences.
- Medium grained : across cells of the dynamic programming table.

Compute Q', QM, Q

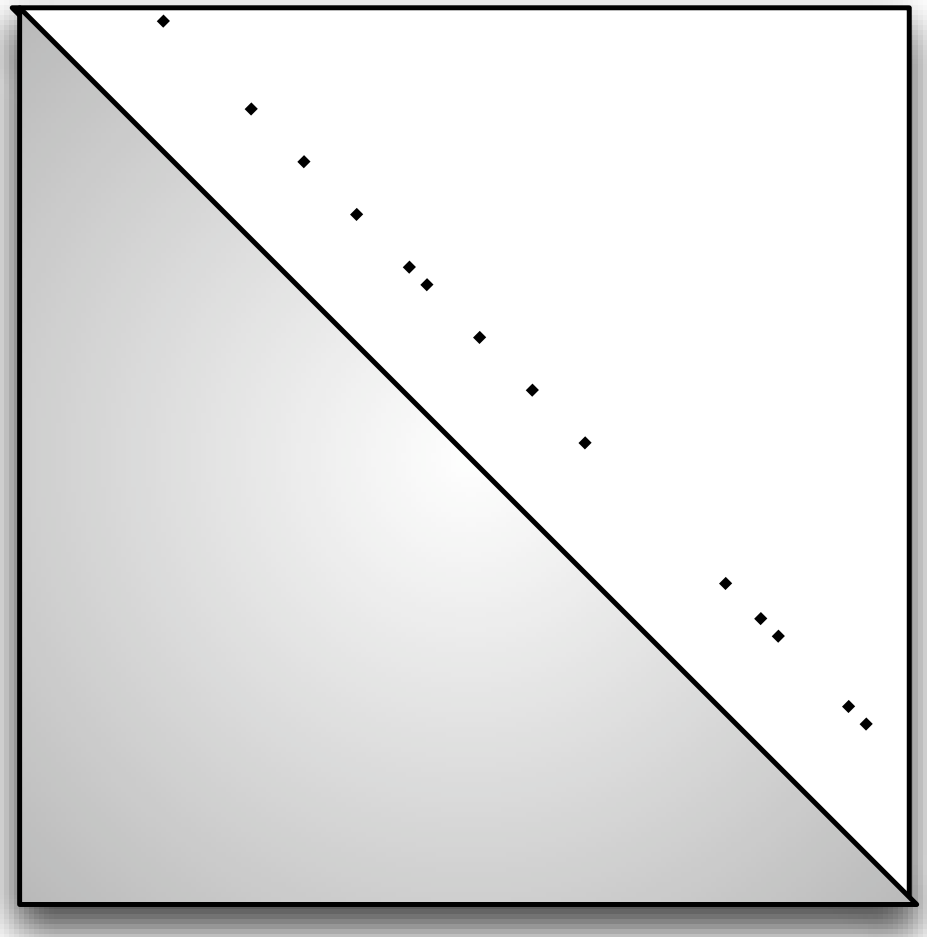


Compute Q', QM, Q

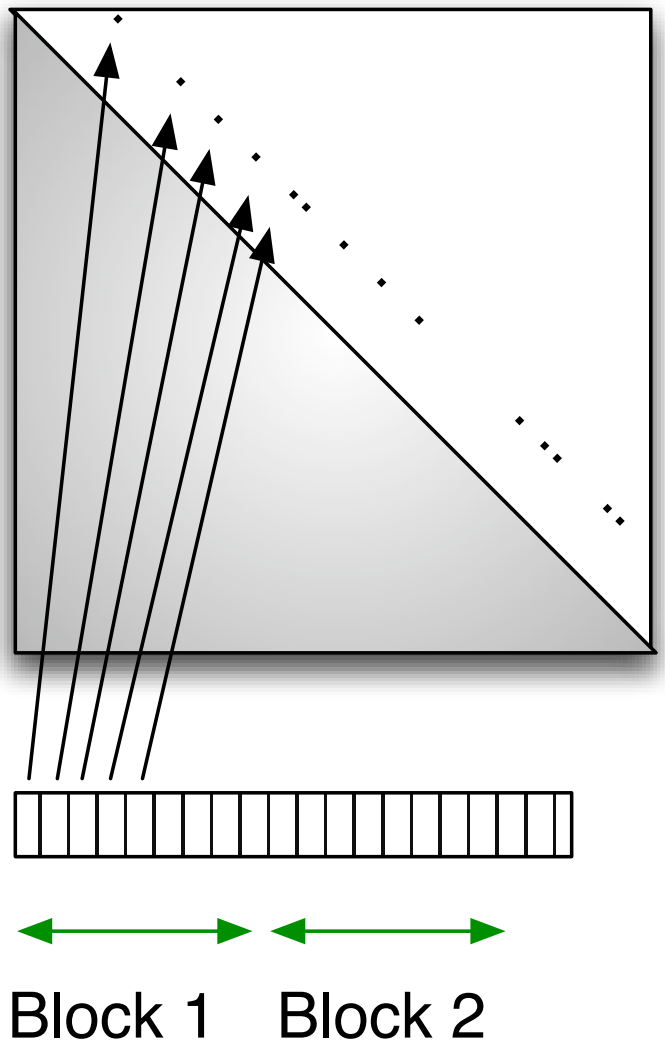


Issue 1 : Divergence

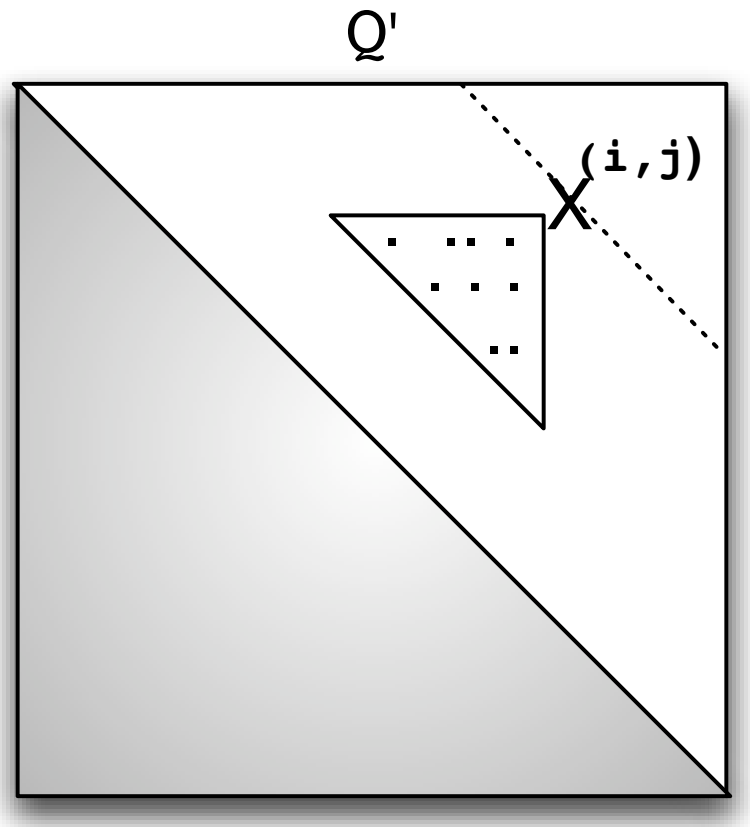
Allowed pairs : A-U C-G G-U

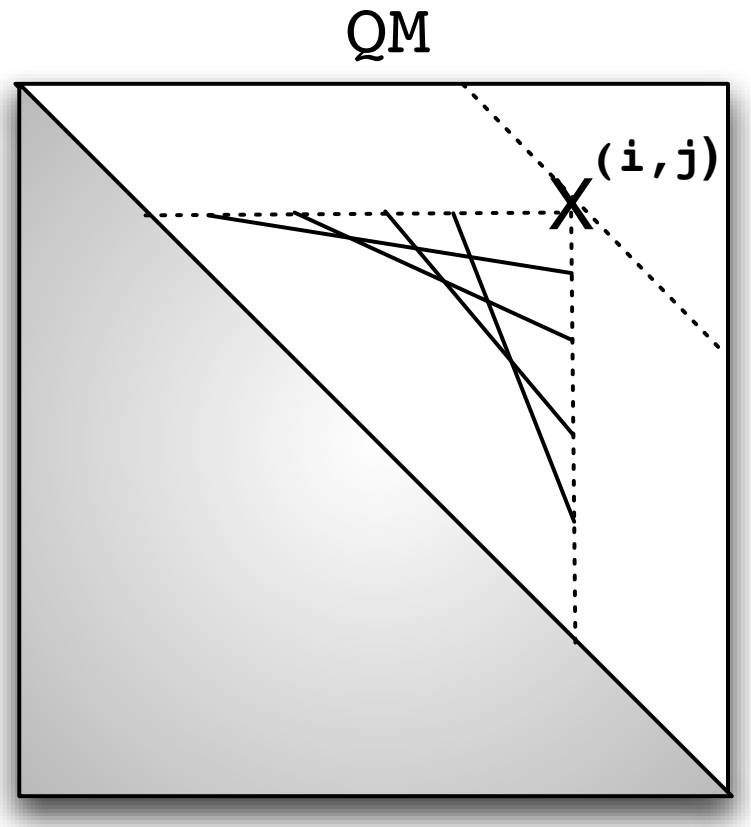


Solution : index computed on CPU



Issue 2 : Complex memory pattern





$$\min_{k,l \in]i;j[^2} Ei(i,j,k,l) + Q'_{k,l}$$

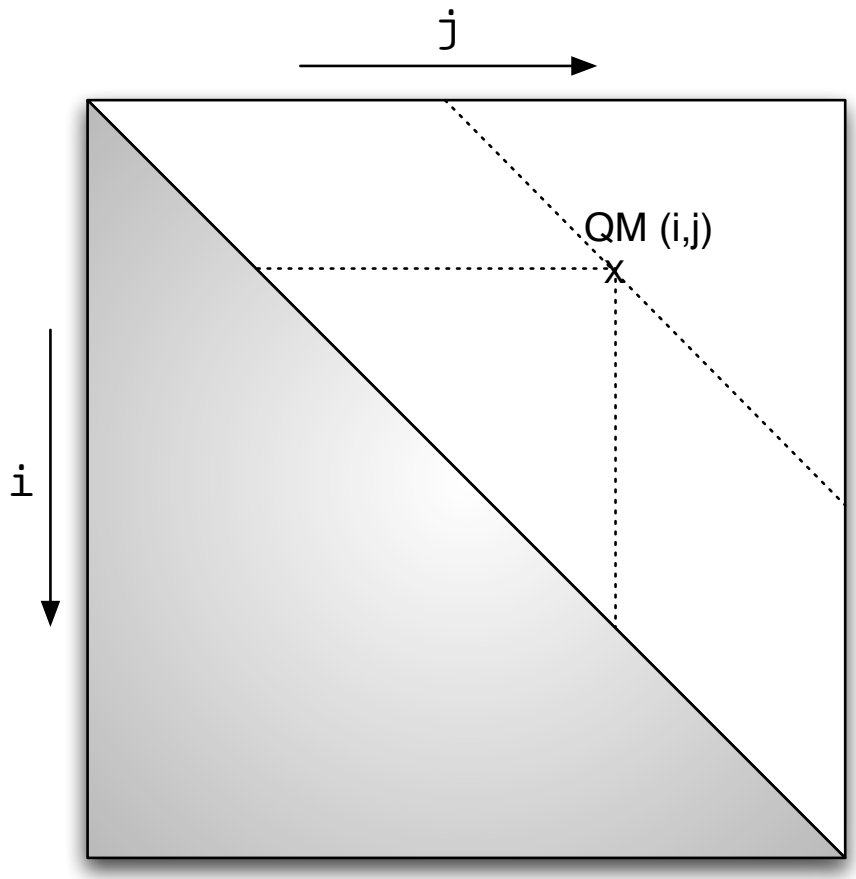
$$\min_{i<k<j} \left(Q_{i,k} + Q_{k+1,j}\right)$$

Complex memory pattern

- Un-coalesced memory access.
- Too big for shared memory.
- $\longrightarrow$ Optimization of register usage to hide latency.
- $\longrightarrow$ Exploration of many block/threads configurations.
- $\longrightarrow$ Fine- tuning usage of memory spaces : texture, constant, global.

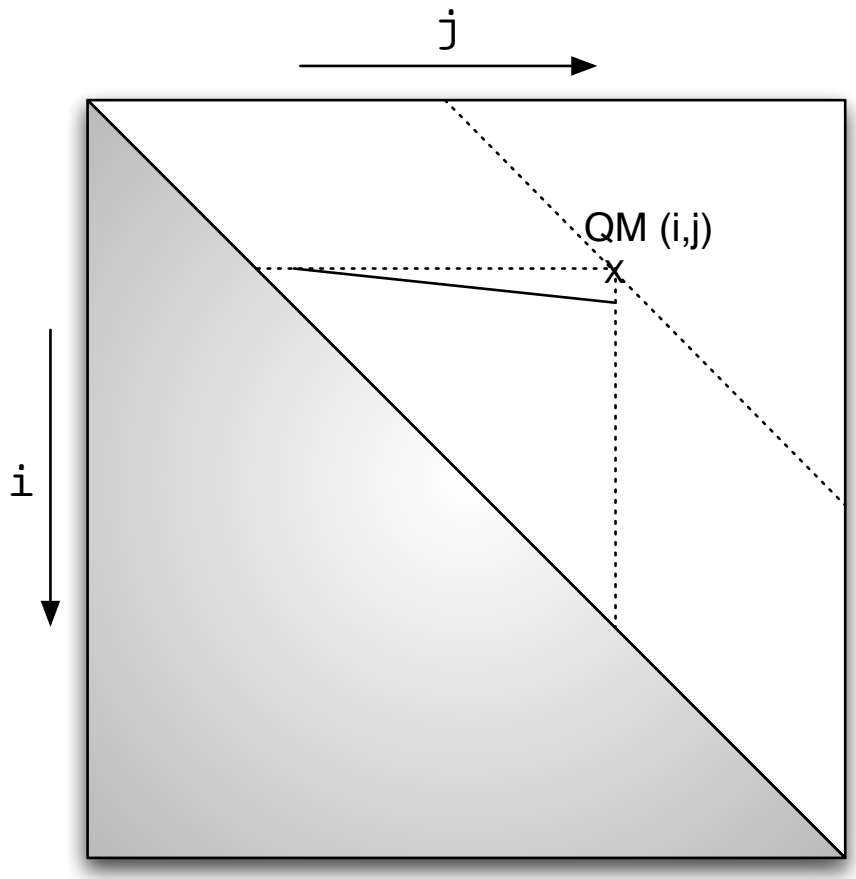
Data reuse problem

$$QM_{i,j} = \min_{i<k<j} (Q_{i,k} + Q_{k+1,j})$$



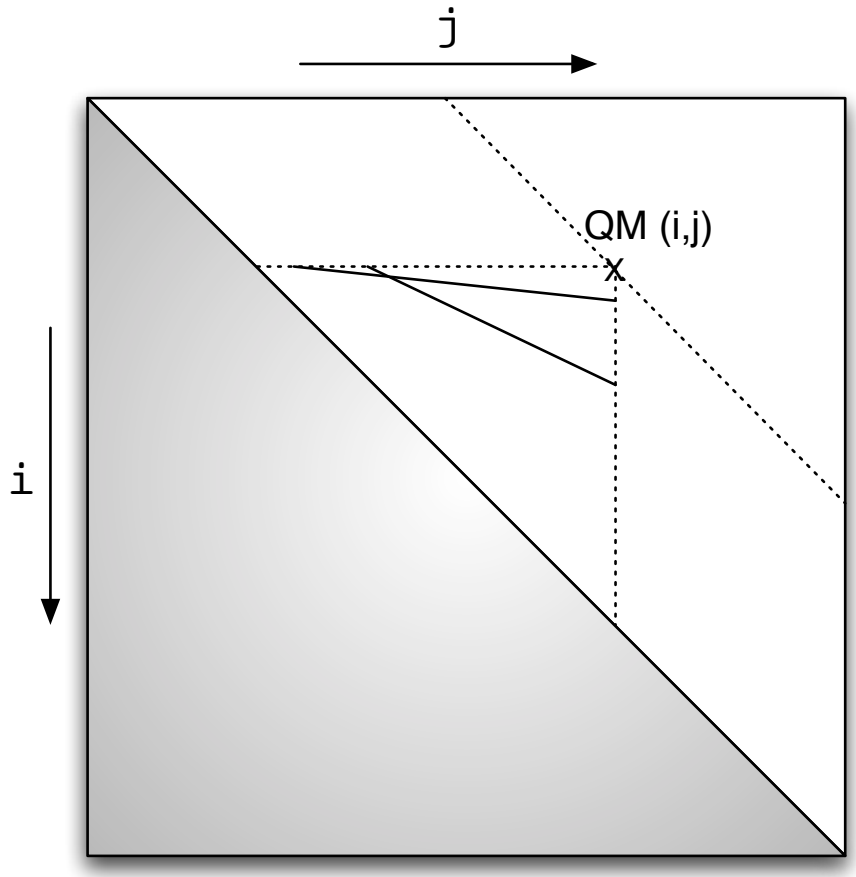
Data reuse problem

$$QM_{i,j} = \min_{i<k<j} (Q_{i,k} + Q_{k+1,j})$$



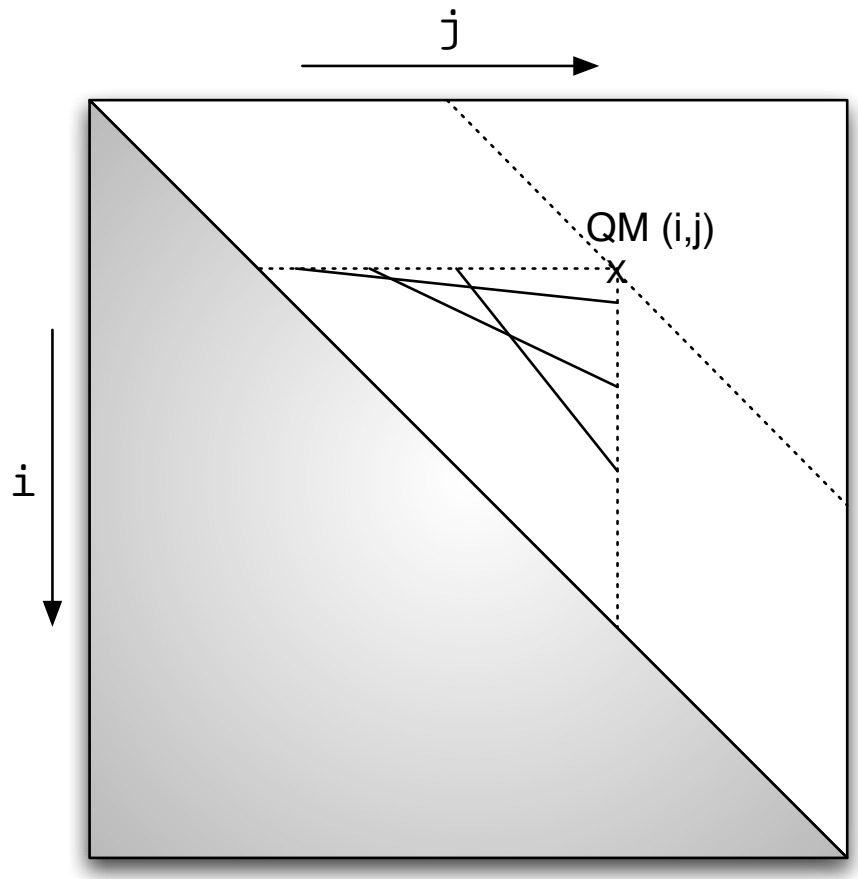
Data reuse problem

$$QM_{i,j} = \min_{i<k<j} (Q_{i,k} + Q_{k+1,j})$$



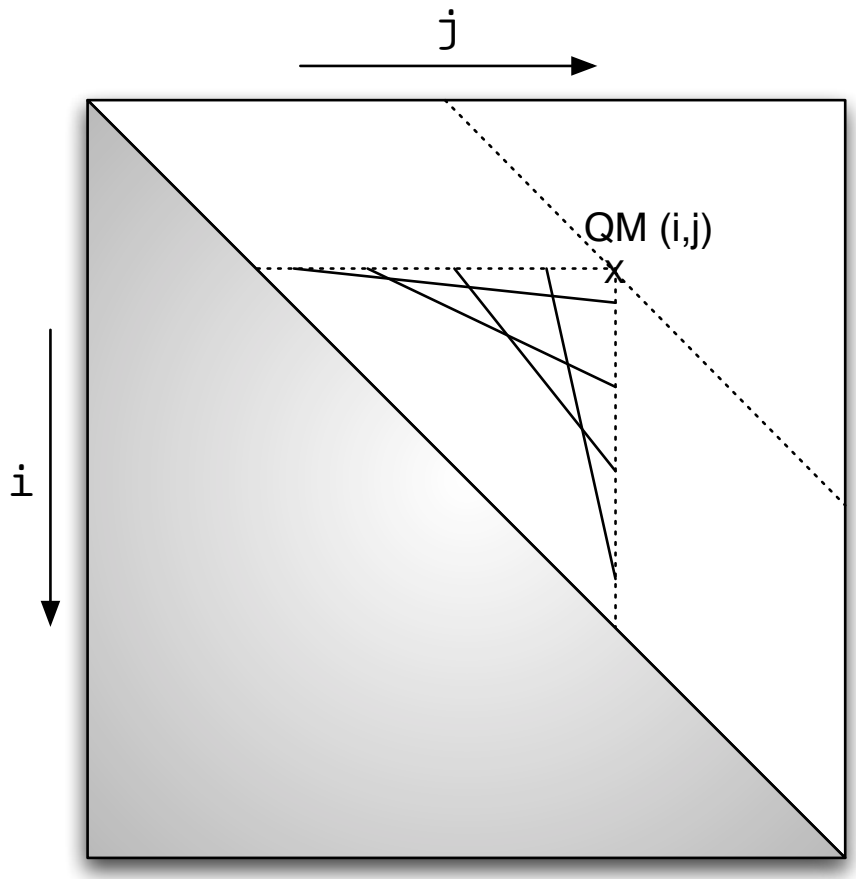
Data reuse problem

$$QM_{i,j} = \min_{i<k<j} (Q_{i,k} + Q_{k+1,j})$$



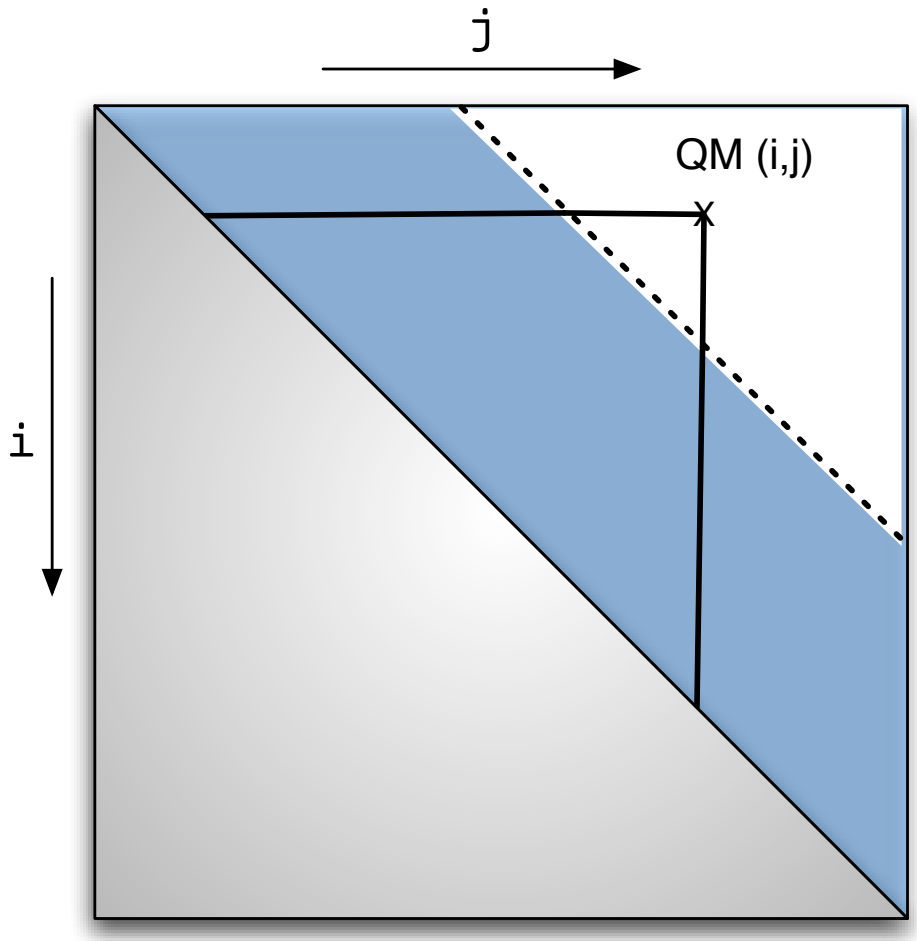
Data reuse problem

$$QM_{i,j} = \min_{i<k<j} (Q_{i,k} + Q_{k+1,j})$$



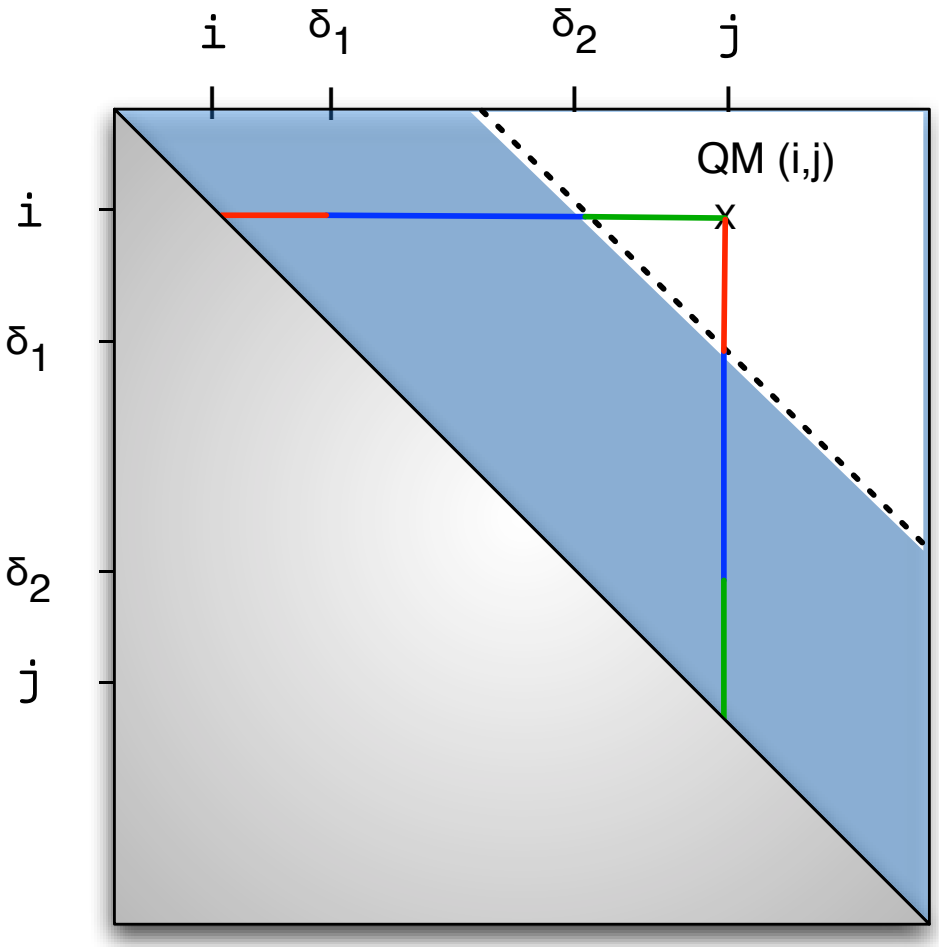
Reduction split

$$QM_{i,j} = \min_{i<k<j} (Q_{i,k} + Q_{k+1,j})$$



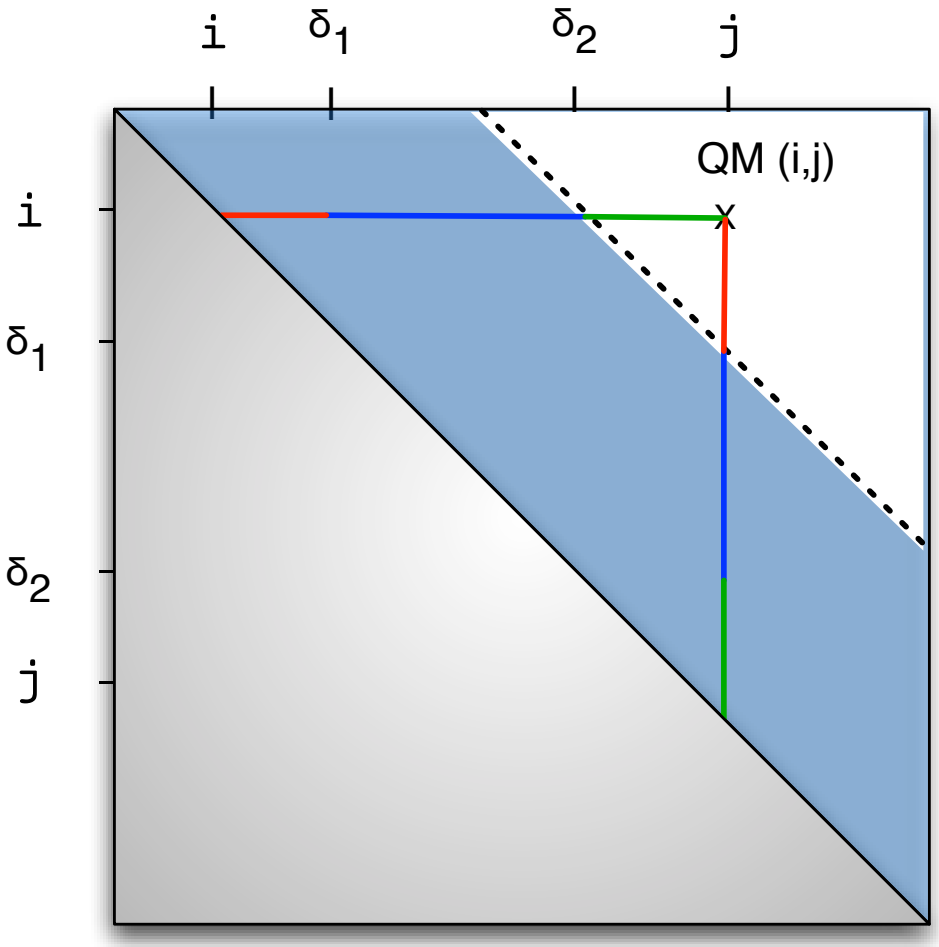
Reduction split

$$QM_{i,j} = \min_{i<k<j}(Q_{i,k} + Q_{k+1,j}) = \min(\min_{i<k<\delta_1}, \min_{\delta_1<k<\delta_2}, \min_{\delta_2<k<j})$$



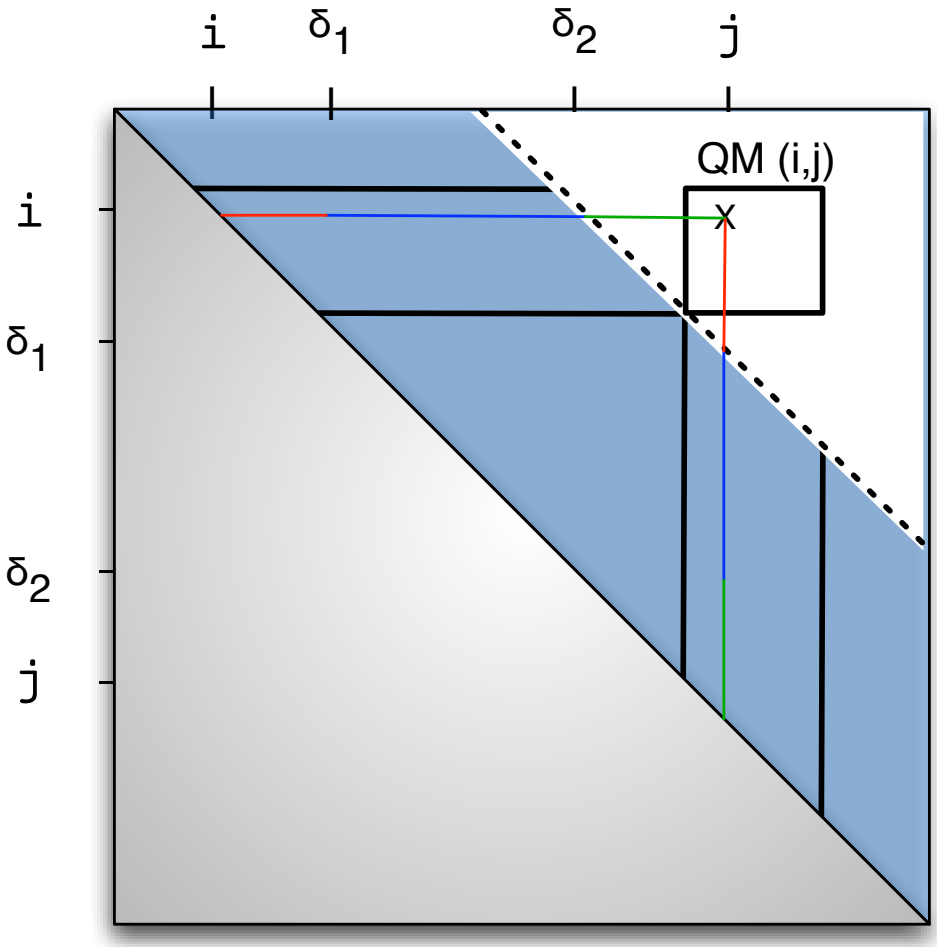
Reduction split

$$QM_{i,j} = \min_{i<k<j}(Q_{i,k} + Q_{k+1,j}) = \min(\min_{\delta_1<k<\delta_2}, \min_{i<k<\delta_1}, \min_{\delta_2<k<j})$$



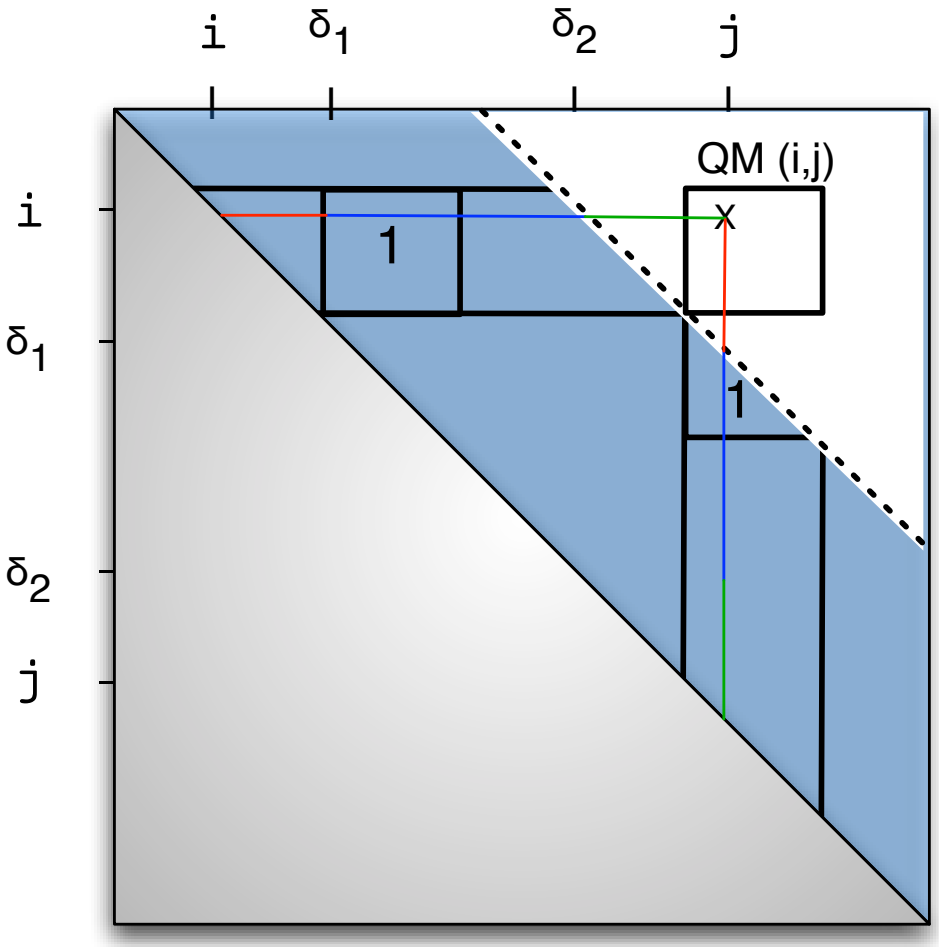
Tiling

$$QM_{i,j} = \min_{i<k<j}(Q_{i,k} + Q_{k+1,j}) = \min(\min_{\delta_1<k<\delta_2}, \min_{i<k<\delta_1}, \min_{\delta_2<k<j})$$



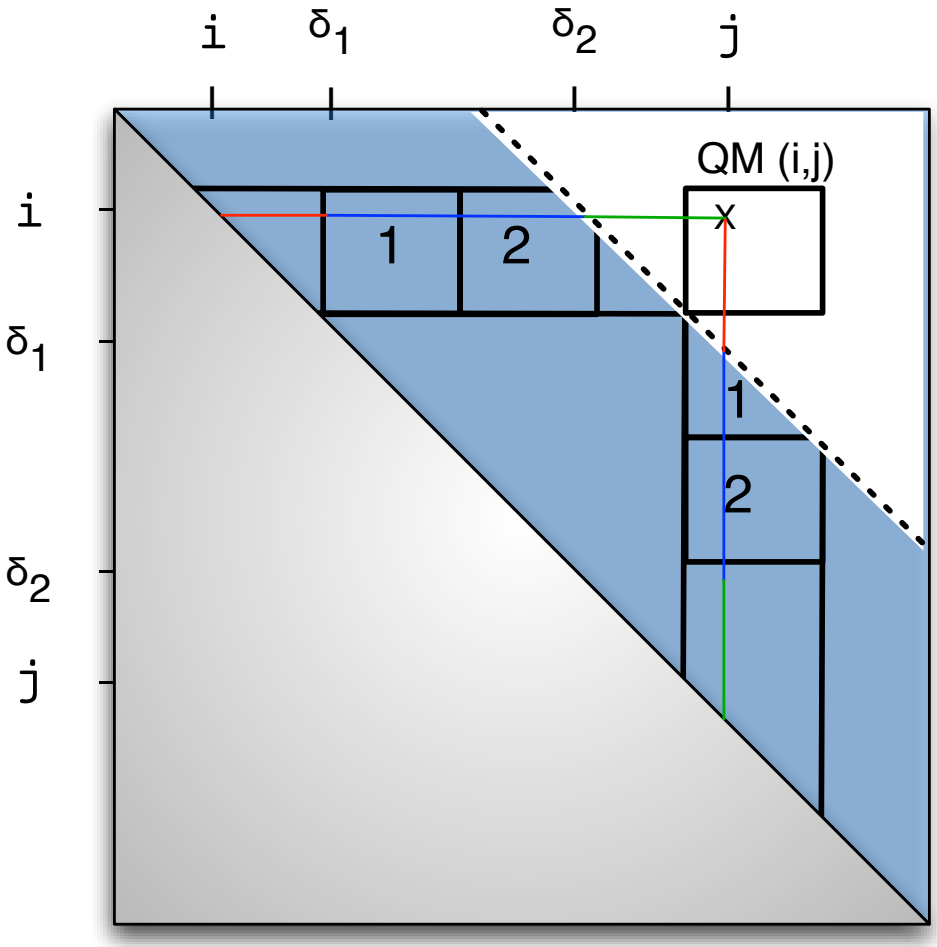
Tiling

$$QM_{i,j} = \min_{i<k<j}(Q_{i,k} + Q_{k+1,j}) = \min(\min_{\delta_1<k<\delta_2}, \min_{i<k<\delta_1}, \min_{\delta_2<k<j})$$

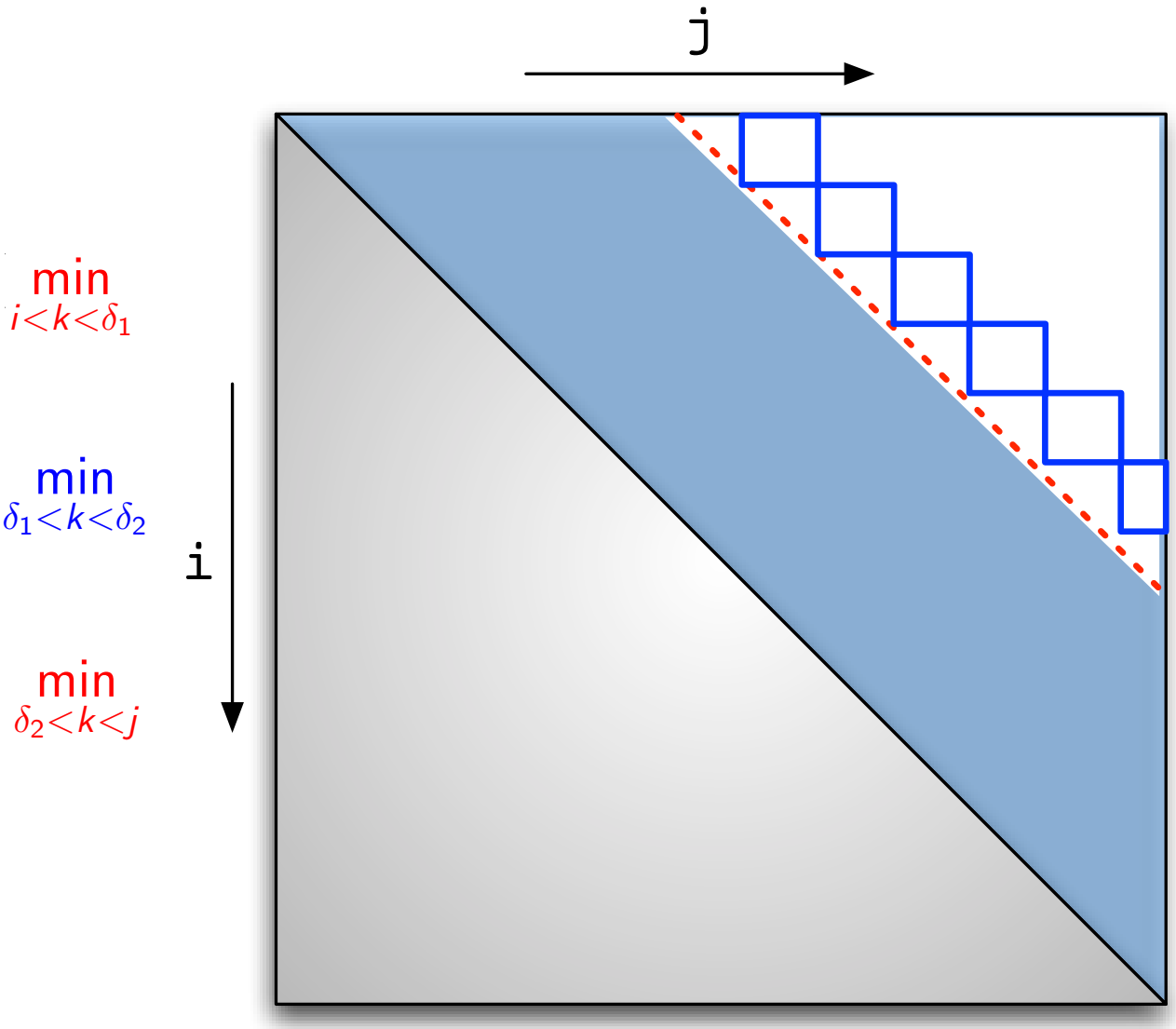


Tiling

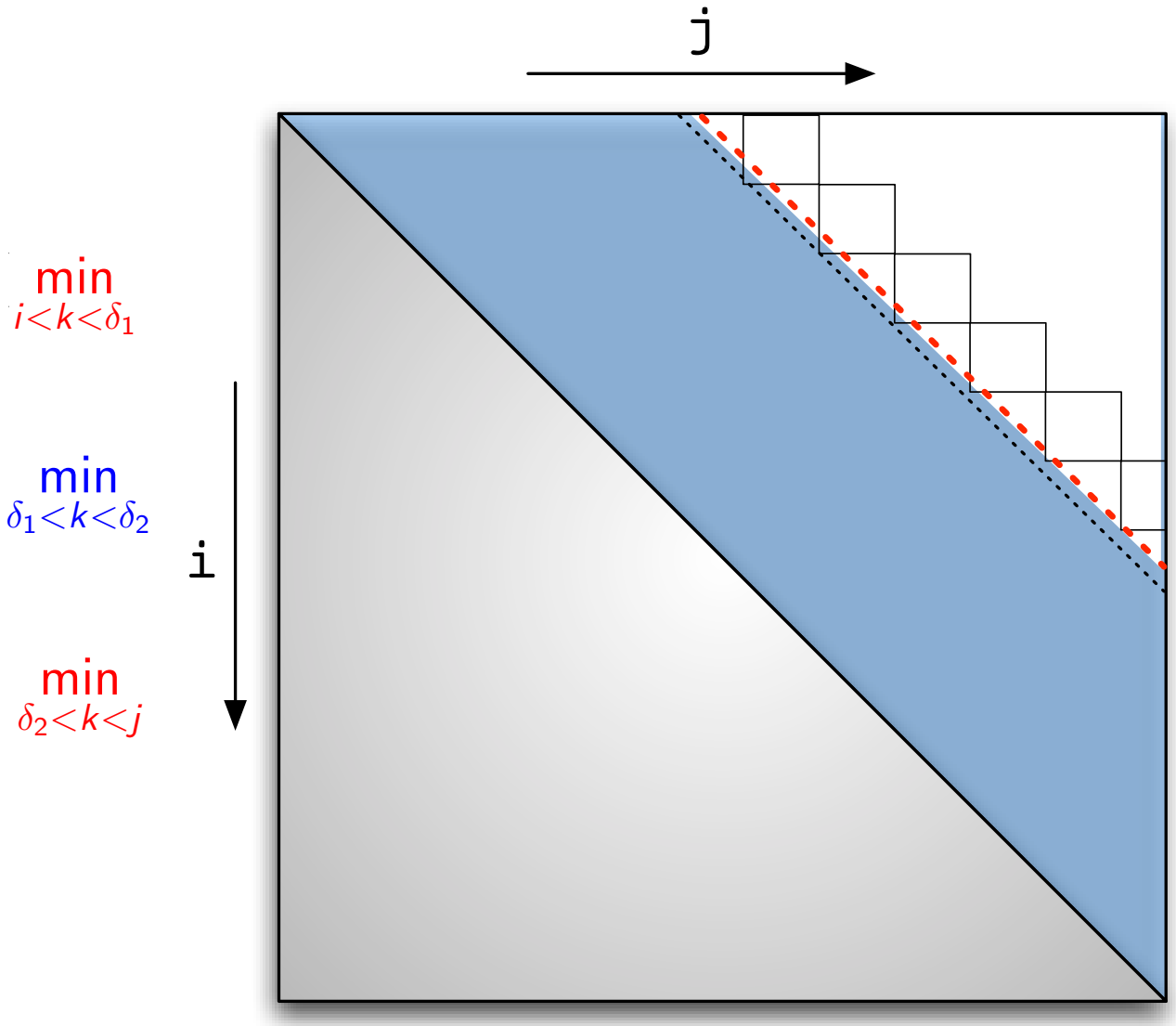
$$QM_{i,j} = \min_{i<k<j}(Q_{i,k} + Q_{k+1,j}) = \min(\min_{\delta_1<k<\delta_2}, \min_{i<k<\delta_1}, \min_{\delta_2<k<j})$$



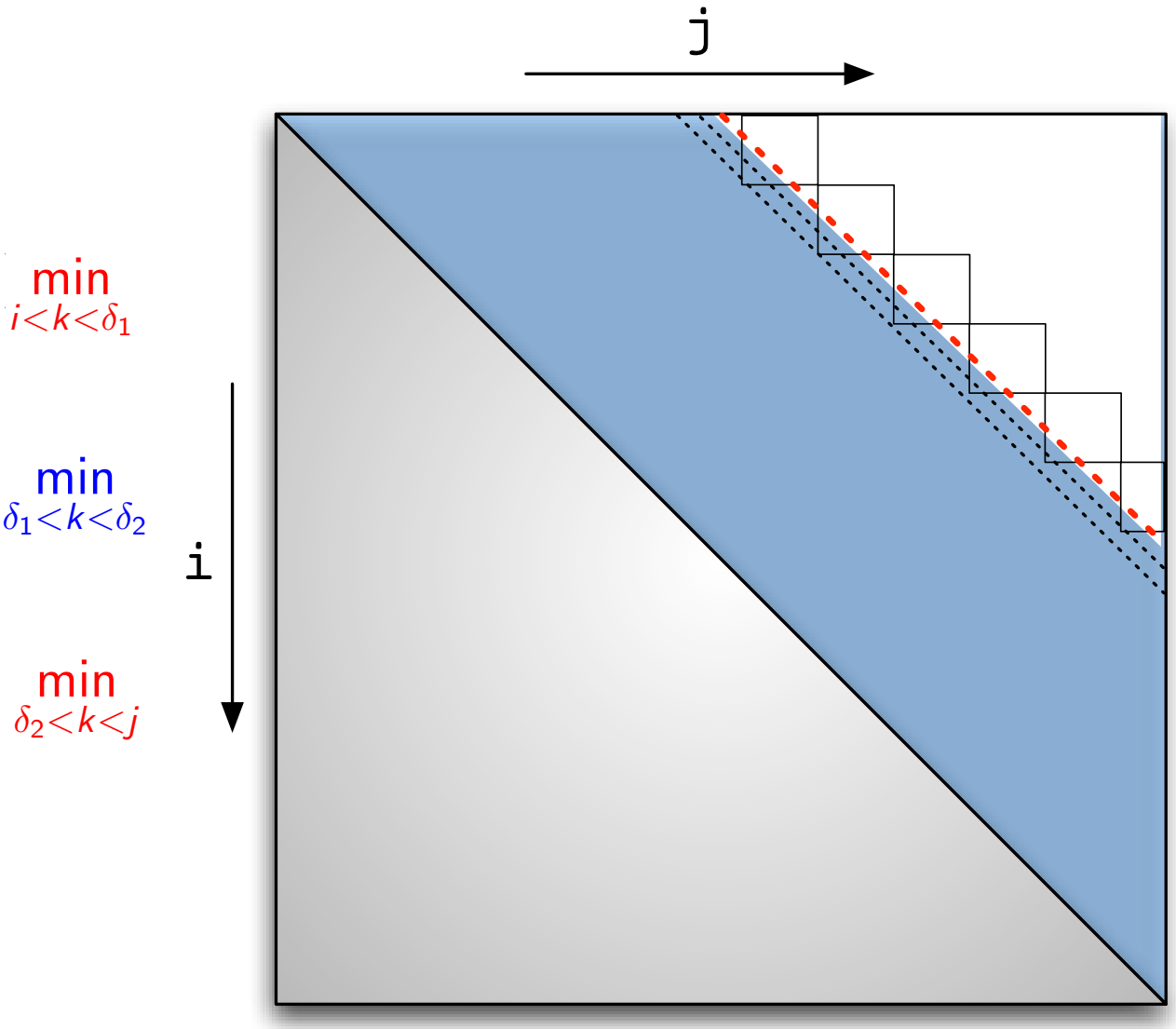
Tiling : Big picture



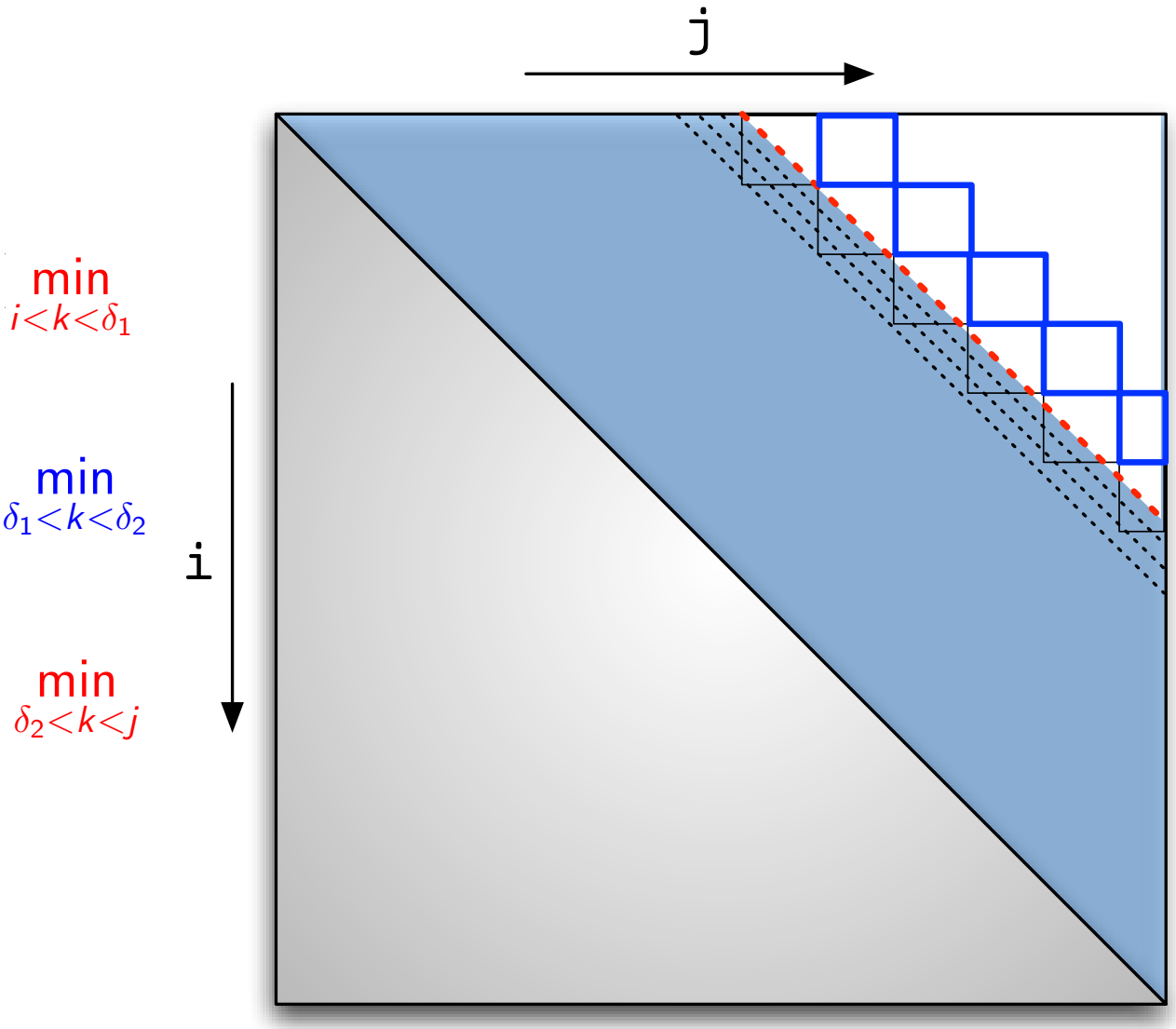
Tiling : Big picture



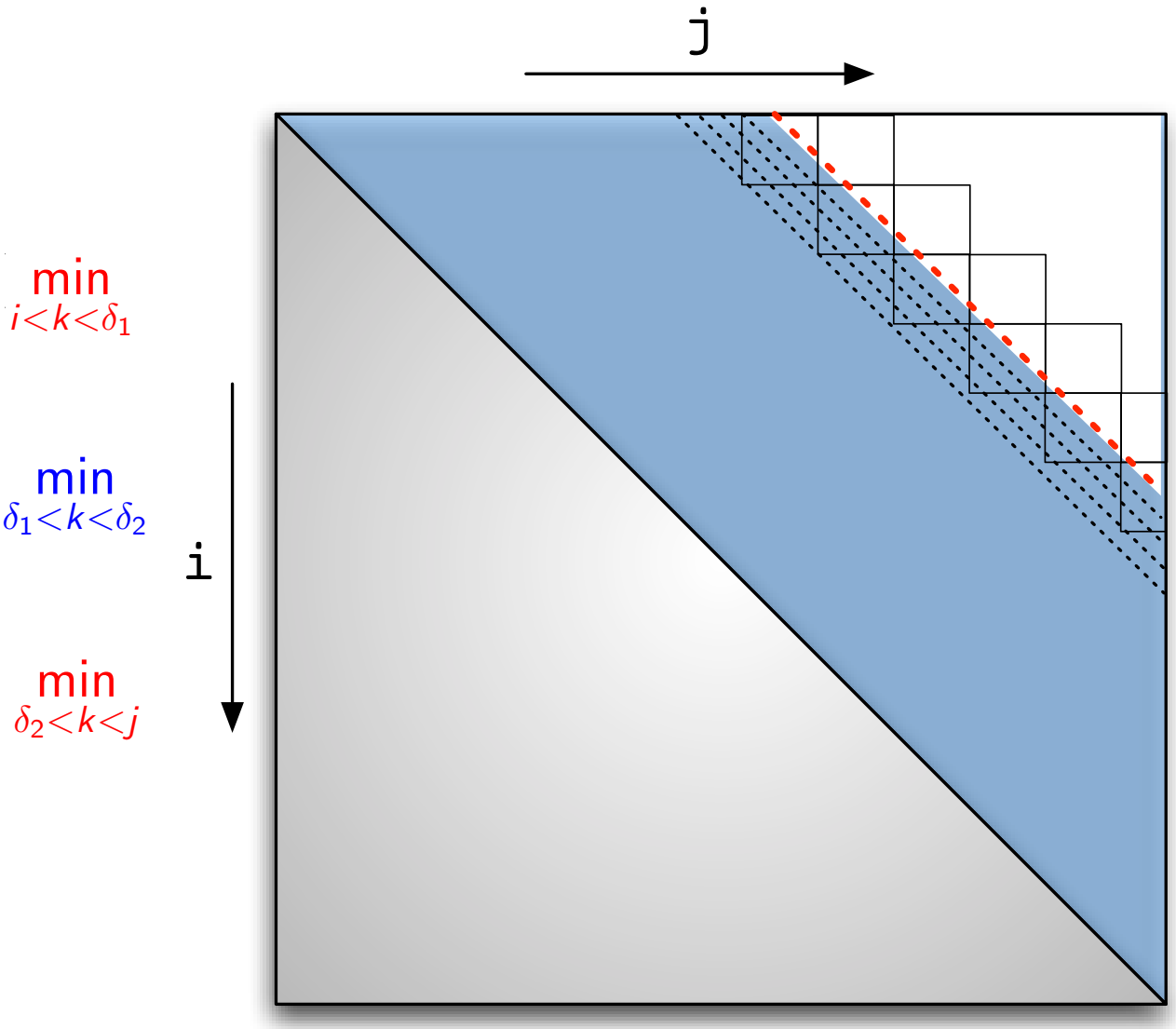
Tiling : Big picture



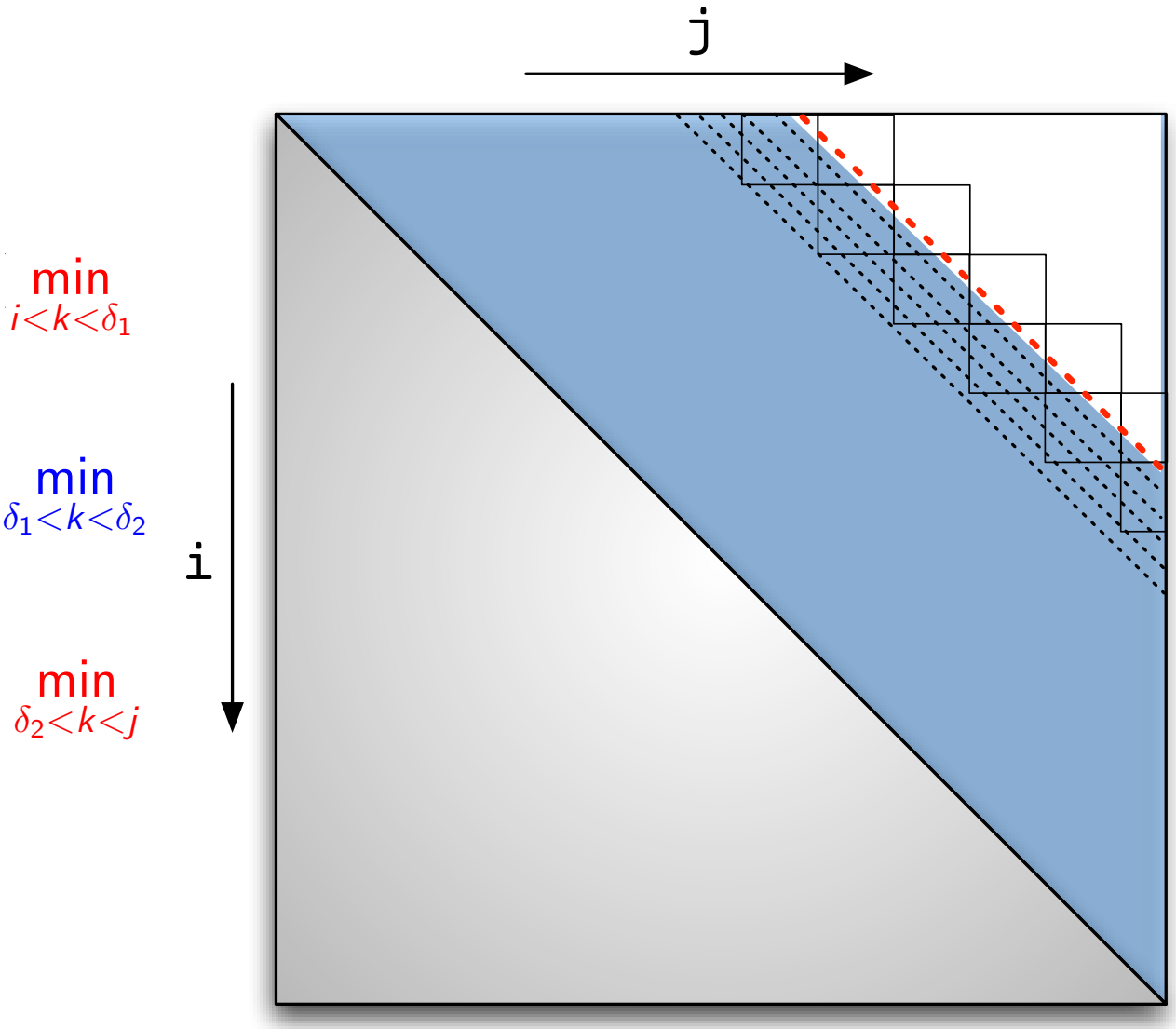
Tiling : Big picture



Tiling : Big picture



Tiling : Big picture

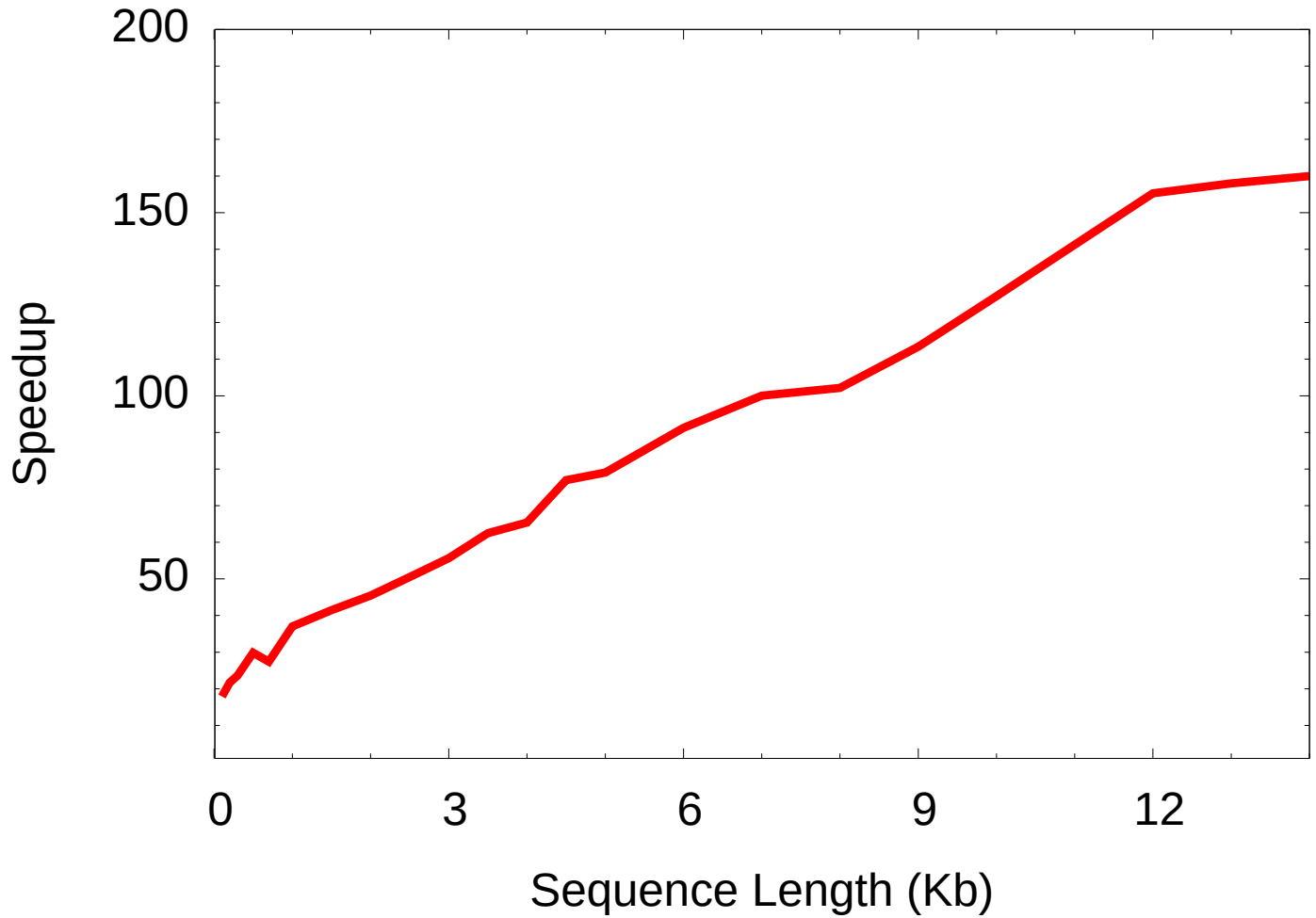


Applications

- Search miRNAs genome-wide : folding of millions of 120-nt sequences [Stark et al., Gen. Research 2007] .
- Structures of 11 picornaviral RNA sequences [Palmenberg and Sgro, Sem. Virol. 1997]
- Statistical study of DNA sequences, needs folding of thousands of randomized sequences.

GPU speedup vs Unafold CPU

1 Tesla C1060 vs Xeon E5450 3.00 Ghz (1 core)

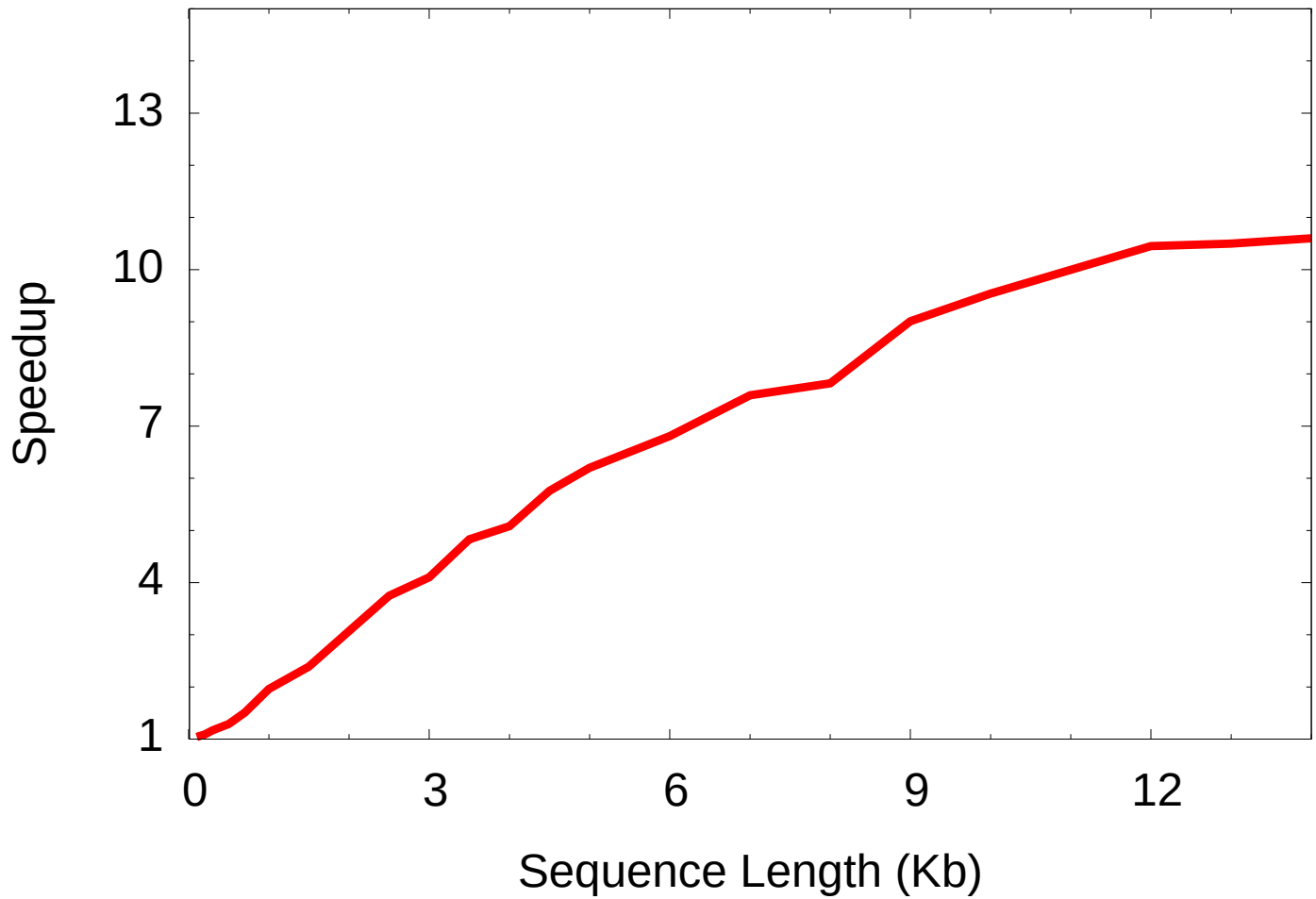


CPU optimization

- Same Tiling scheme on sequential CPU program.
- Better cache locality.
- Provides easy SSE-vectorization.
- Second level of tiling : reuse data loaded in SSE registers.

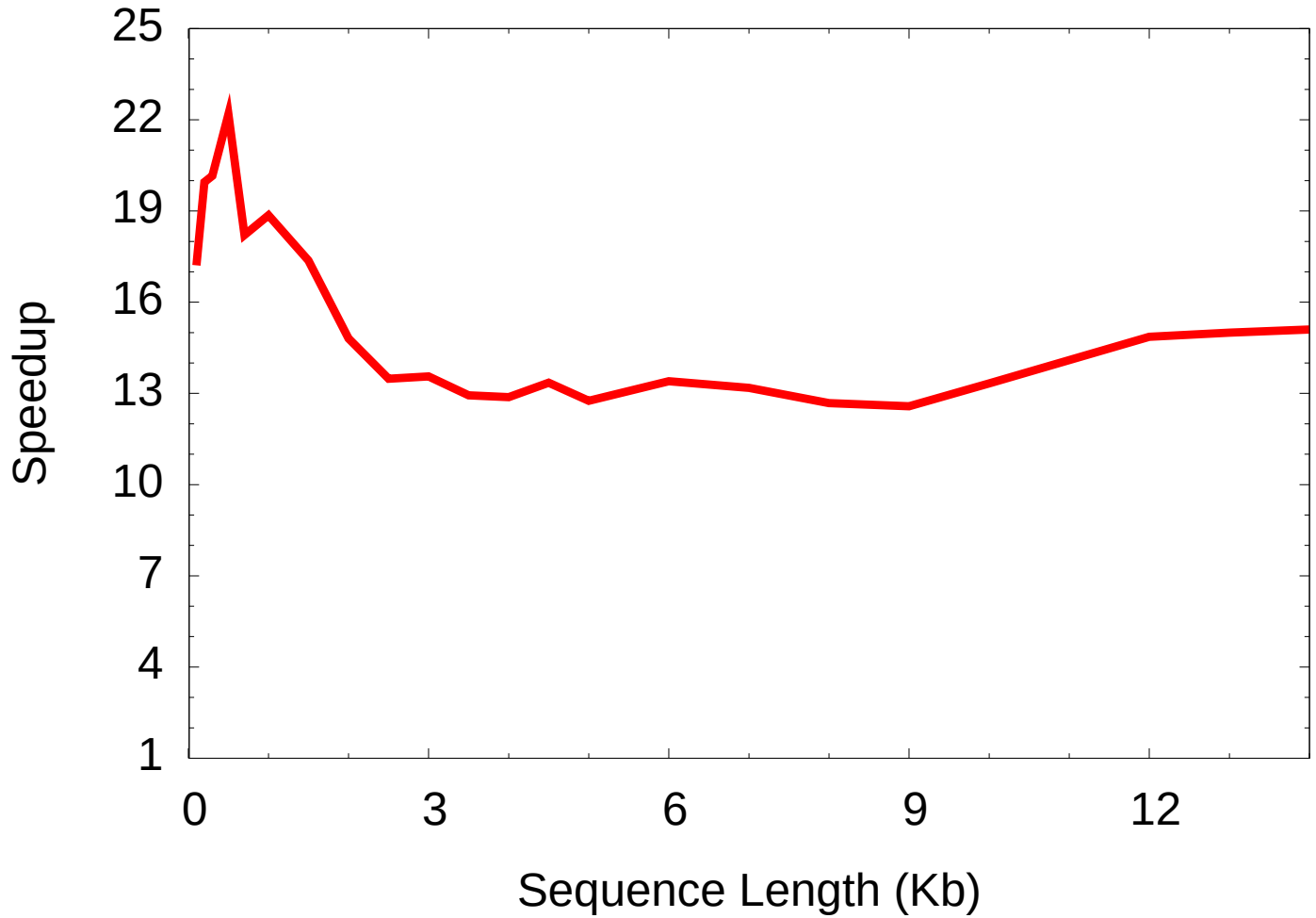
CPU-tiled +SSE speedup vs Unafold CPU

On Xeon E5450 3.00 Ghz (1 core)

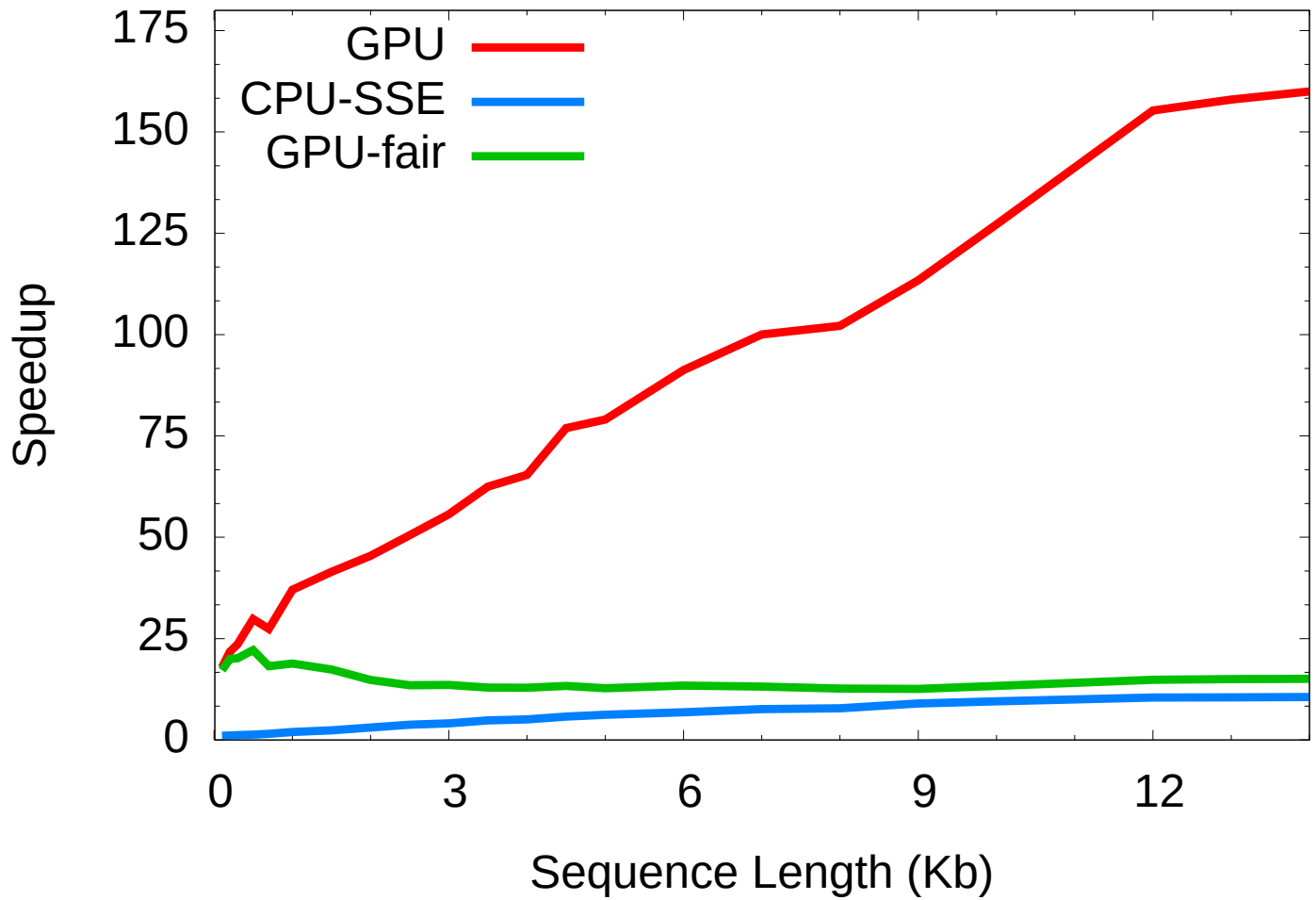


Fair GPU-CPU comparison

1 Tesla C1060 vs Xeon E5450 3.00 Ghz (1 core)



Fair GPU-CPU comparison

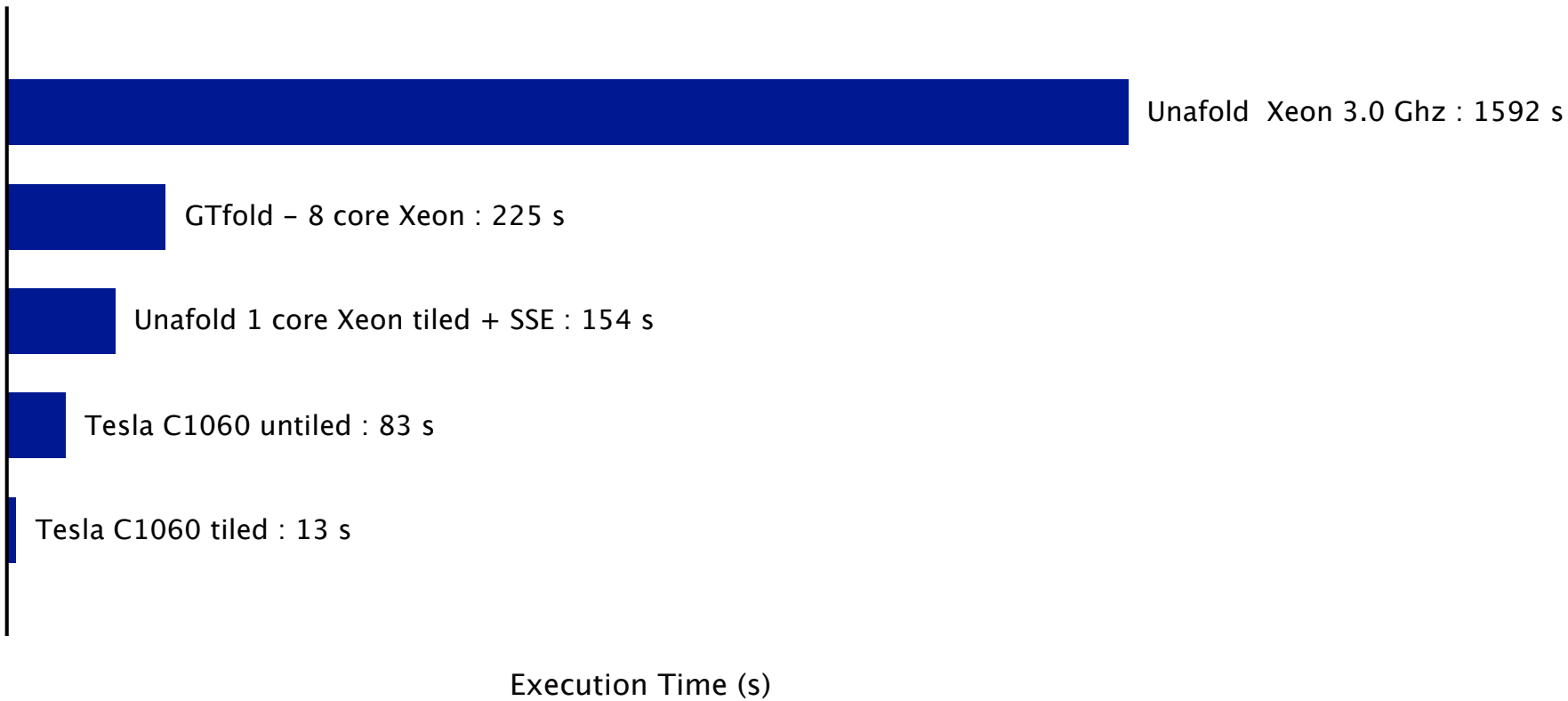


Related work : GTfold

OpenMP Parallelization on multi-core CPU.
Parallelization across points of a diagonal.

Mathuriya A., Bader D.A., Heitsch C.E, and Harvey S.C : GTfold : A Scalable Multicore Code for RNA Secondary Structure Prediction", SAC, Computational Sciences Track, Honolulu, March 8-12, 2009

On a 9781 nt virus sequence



Conclusion and perspectives

- Exploiting data reuse is the most important.
- CPU optimization for fair comparison.
- Here, GPU power efficiency $\approx$ CPU power efficiency.
- Many algorithms variations to explore (partition function, suboptimal structures).

Preliminary Power analysis

- GPU Versus 1-core Xeon, SSE optimized implementation : $\times 12$ speedup.
- Tesla C1060 : 190 W TDP.
- 4-core Xeon E5450 3.0 Ghz : 80W TDP.
- $190/(3 \times 80) = 0.8$
- Here GPU is only 20% more power efficient.

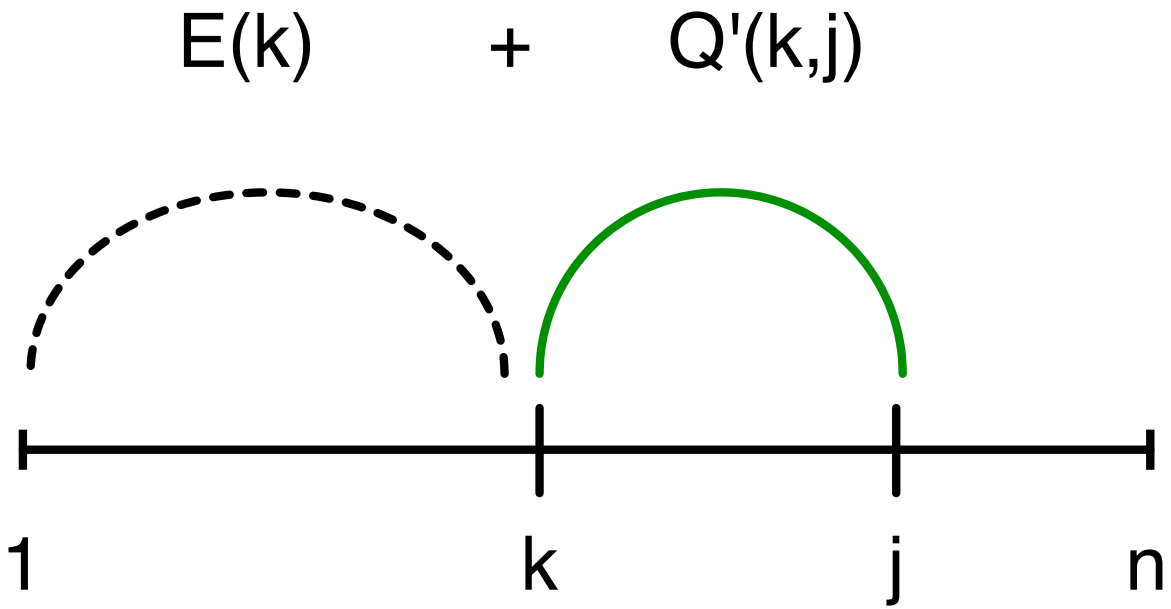
Dynamic Programming

3 Tables representing minimum energy of subsequence i, j :

- $Q'_{i,j}$: i and j are paired.
- $Q_{i,j}$: with at least one pair, inside a multiloop.
- $QM_{i,j}$: with at least two pairs , inside a multiloop.

Decomposition :

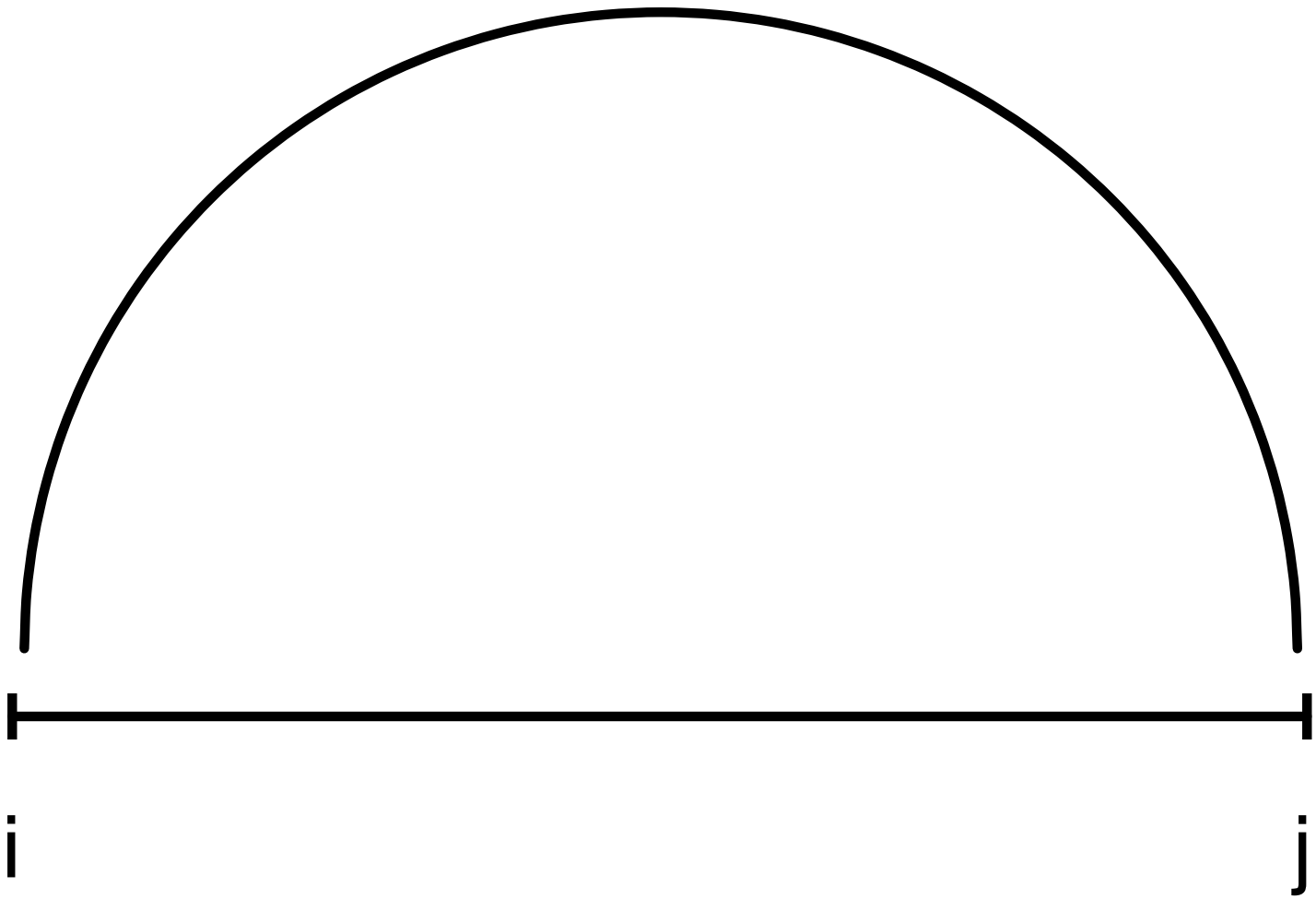
$E(k)$ min energy of subsequence $1..k$



$$E_j = \min \begin{cases} E_{j-1} \\ \min\limits_{1<k<j} (E_{k-1} + Q'_{k,j}) \end{cases} \quad (4)$$

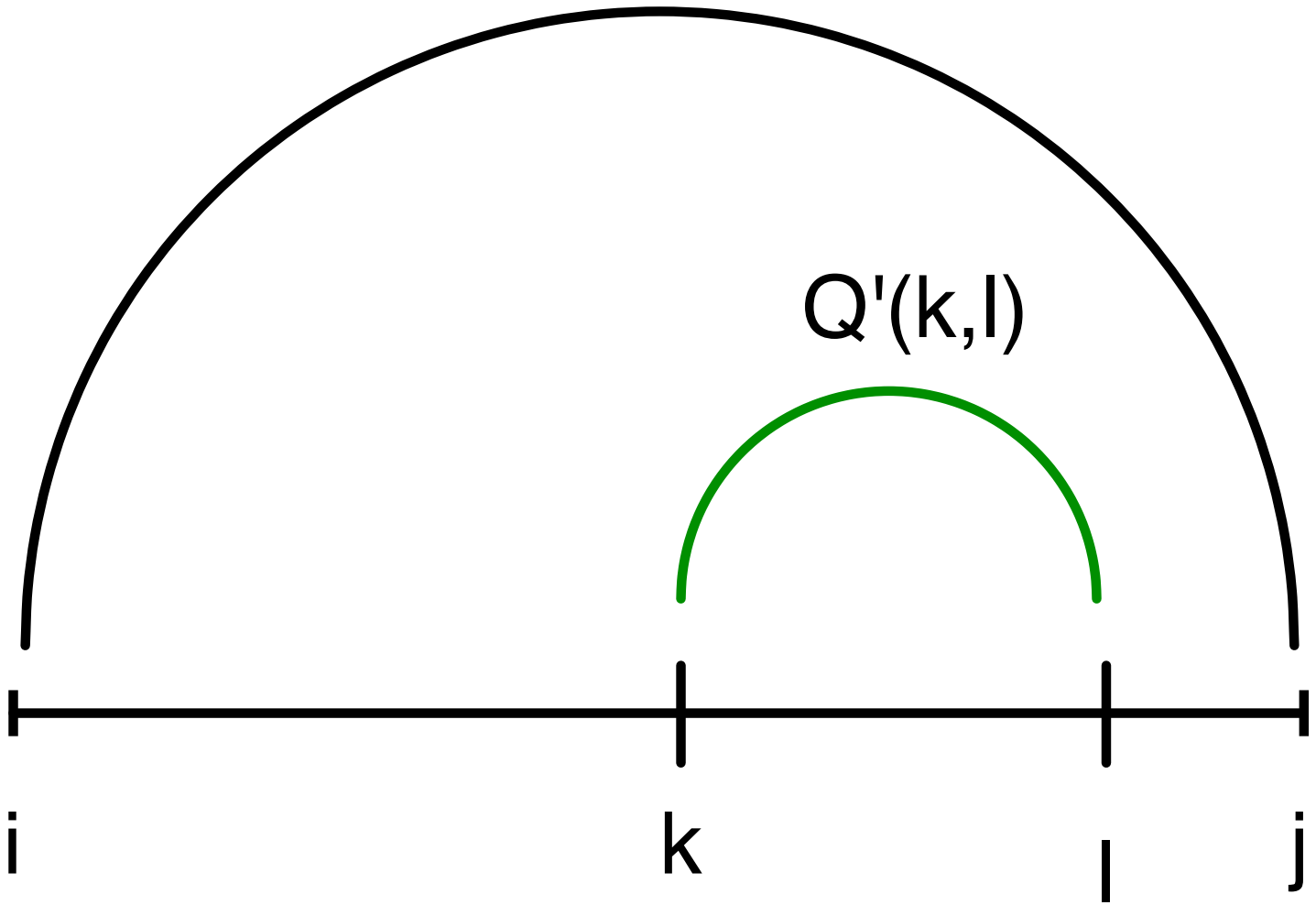
$Q'_{i,j}$ computation

No inside pairs : hairpin.



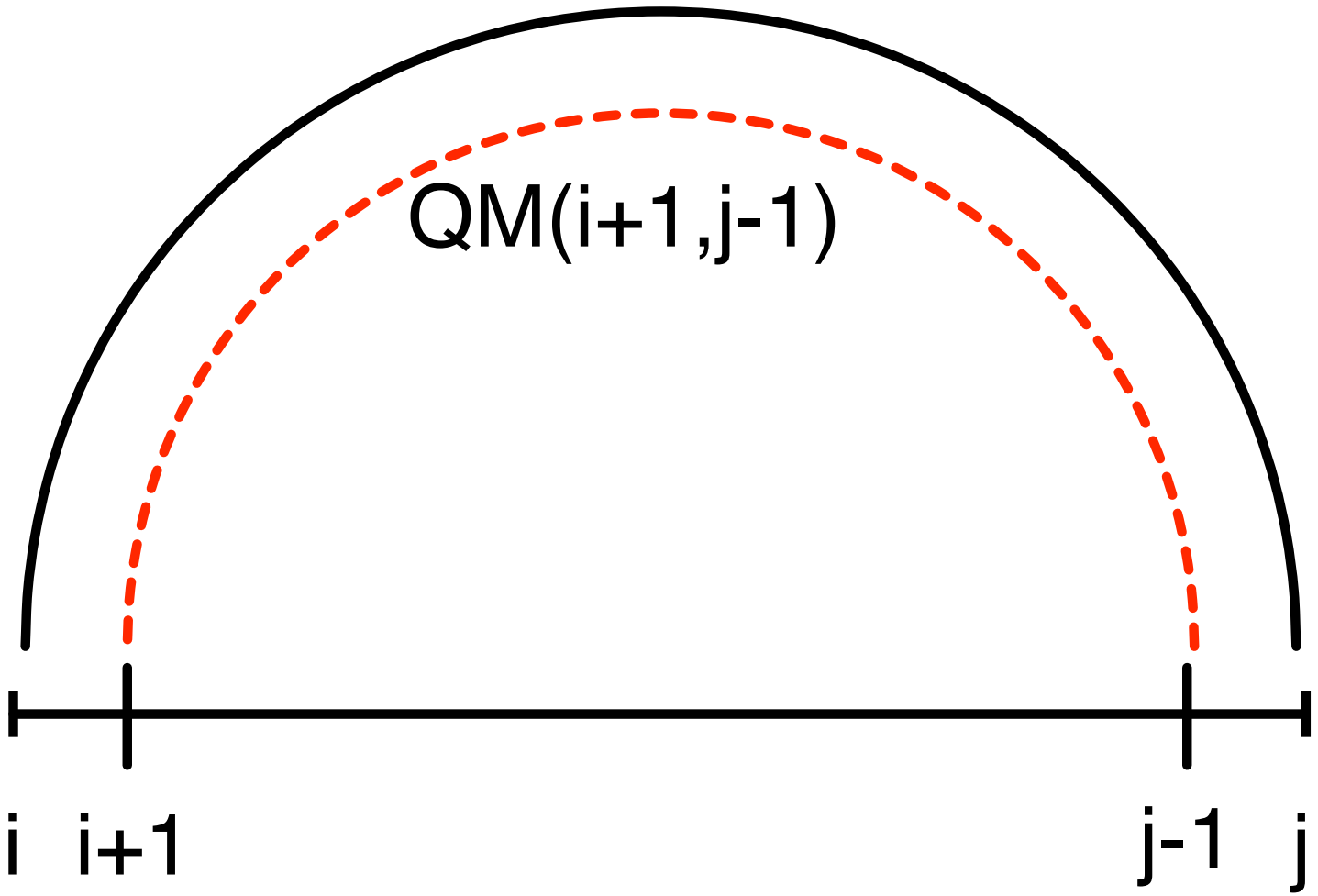
$Q'_{i,j}$ computation

One base pair : internal loop.



$Q'_{i,j}$ computation

Several base pairs : multi-loop.

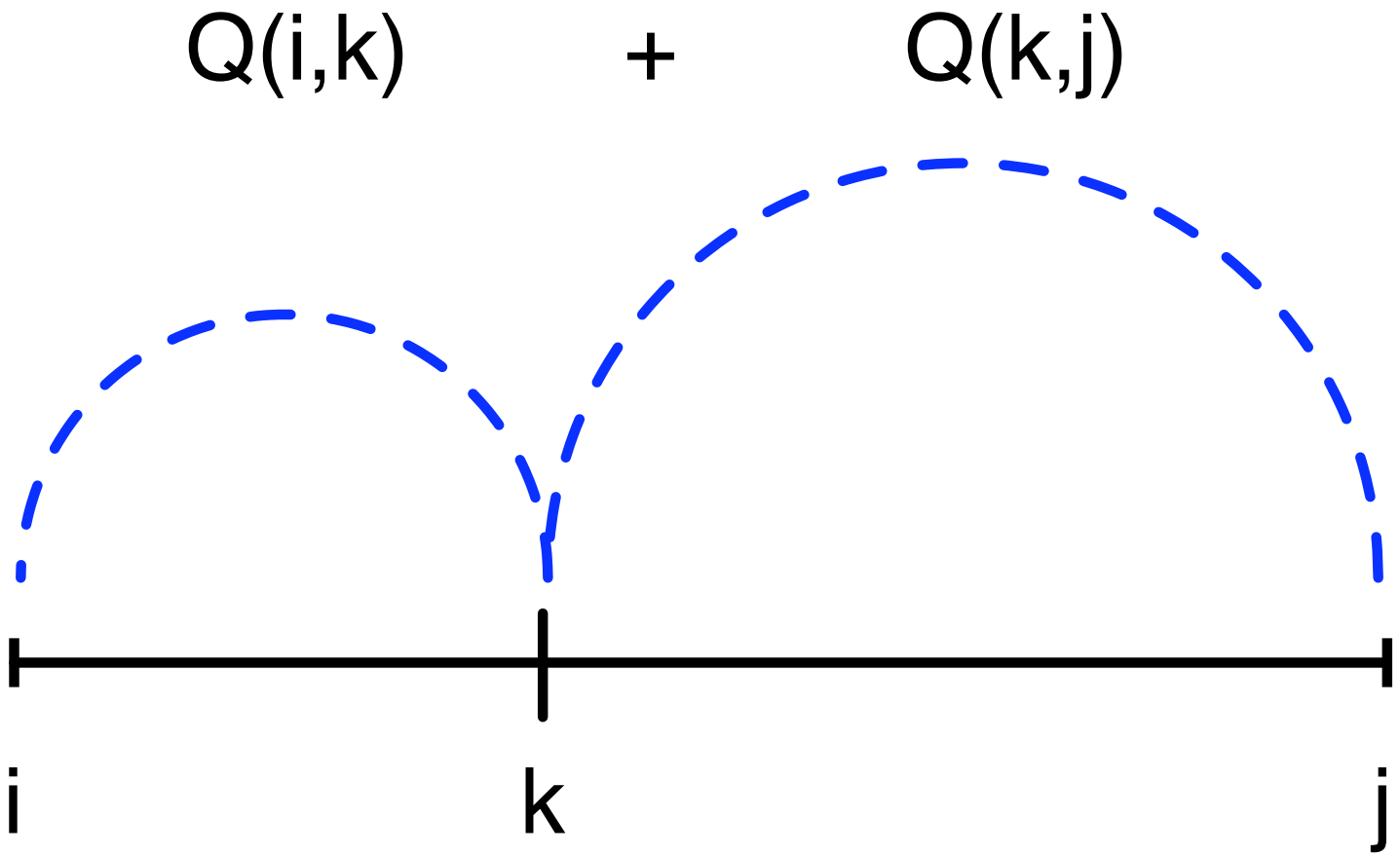


$Q'_{i,j}$ formula

$$Q'_{i,j} = \begin{cases} \min \begin{cases} Eh(i,j) \\ Es(i,j) + Q'_{i+1,j-1} \\ \min\limits_{k,l \in]i;j[^2} Ei(i,j,k,l) + Q'_{k,l} \\ QM_{i+1,j-1} \end{cases} & i \cdot j \text{ allowed} \\ \infty & i \cdot j \text{ not allowed} \end{cases} \quad (5)$$

$QM_{i,j}$ computation

QM : at least 2 base pairs inside a multi loop



$Q_{i,j}$ $QM_{i,j}$ formulas

$$QM_{i,j} = \min_{i<k<j} (Q_{i,k} + Q_{k+1,j}) \tag{6}$$

$$Q_{i,j} = \min \begin{cases} QM_{i,j} \\ \min(Q_{i+1,j}, Q_{i,j-1}) \\ Q'_{i,j} \end{cases} \tag{7}$$