

Report Progetto

SciddicaT

227805 Spagnolo Domenico

Sommario

[1. Abstract 3](#_Toc84459819)

[2. Introduction 3](#_Toc84459820)

[3. Parallel Implementations 5](#_Toc84459821)

[ Straightforward parallelization 5](#_Toc84459822)

[ Tiled parallelization with Halo Cells 6](#_Toc84459823)

[ Tiled parallelization without Halo Cells 6](#_Toc84459824)

[ MPI Parallelization 6](#_Toc84459825)

[4. Computational Performance 7](#_Toc84459826)

[ Monolithic 7](#_Toc84459827)

[**SpeedUp** 7](#_Toc84459828)

[**Plot** 7](#_Toc84459829)

[**Profiling** 8](#_Toc84459830)

[ Grid-Stride 8](#_Toc84459831)

[**SpeedUp** 9](#_Toc84459832)

[**Plot** 9](#_Toc84459833)

[**Profiling** 9](#_Toc84459834)

[ Tiles without halo 10](#_Toc84459835)

[**SpeedUp** 10](#_Toc84459836)

[**Plot** 11](#_Toc84459837)

[**Profiling** 11](#_Toc84459838)

[ Tiles with halo 12](#_Toc84459839)

[**SpeedUp** 12](#_Toc84459840)

[**Plot** 12](#_Toc84459841)

[**Profiling** 13](#_Toc84459842)

[5. Roofline Assessment 14](#_Toc84459843)

[**Specifiche JPDM2** 15](#_Toc84459844)

[**Calcolo Intensità aritmetica e applicazione del modello** 15](#_Toc84459845)

[ Monolithic 15](#_Toc84459846)

[ Grid-Stride 16](#_Toc84459847)

[ Tiles without halo 17](#_Toc84459848)

[ Tiles with halo 18](#_Toc84459849)

[6. Conclusions 19](#_Toc84459850)

# Abstract

Obiettivo principale di questo report è lo sviluppo di diverse versioni parallelizzate di SciddicaT con la libreria CUDA.

# Introduction

SciddicaT è un modello ad uso scientifico per la simulazione di flussi di fluidi non inerziali. Si basa sul *Cellular Automata Computational Paradigm* e sul *Minimization Algorithm of the Differences* (per il calcolo dei flussi tra celle adiacenti). Nonostante la sua semplicità, il modello è in grado di simulare frane non inerziali su superfici topografiche reali, come quella del Tessina in Italia avvenuta nel 1982, la cui configurazione iniziale viene anch’essa fornita insieme al codice sorgente dell’applicativo.

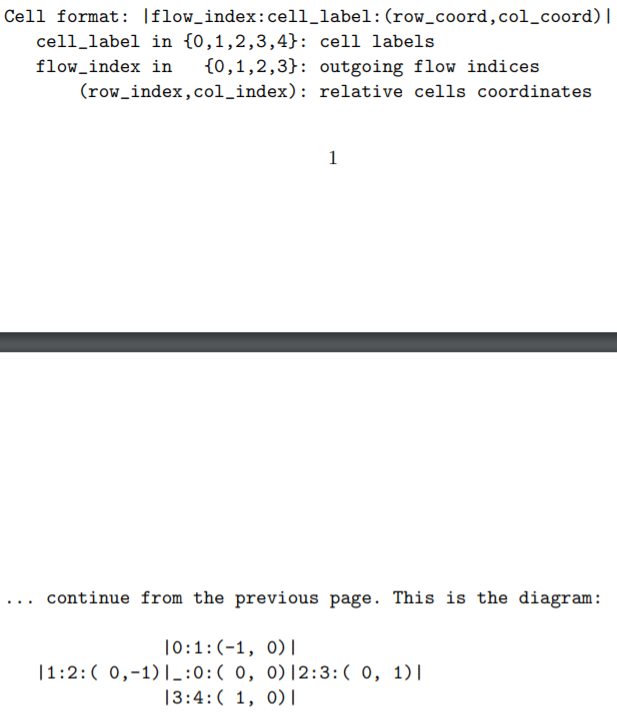
SciddicaT è definito come:

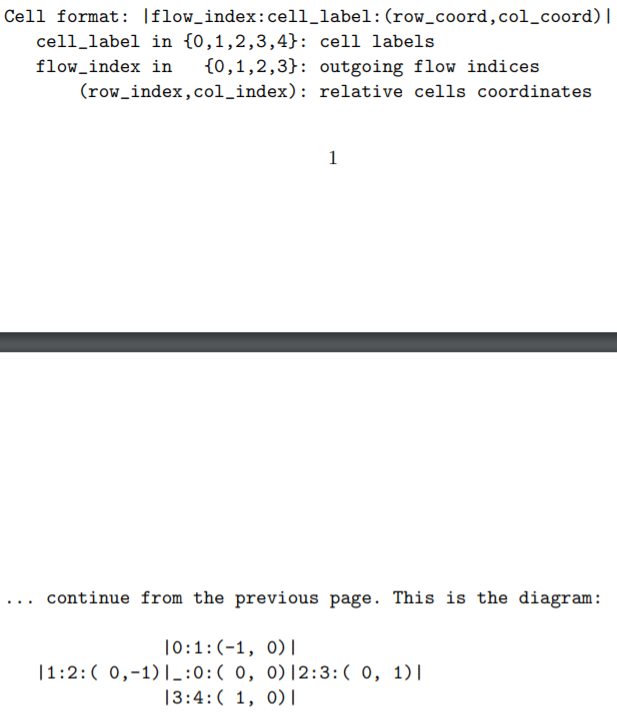
**SciddicaT = <R, X, S, P, σ>**

**R**: è il dominio computazionale bidimensionale, suddiviso in celle di uguale dimensione;

**X**: è l'intorno di von Neumann (un pattern geometrico che identifica la cella centrale e un insieme di quattro celle poste nelle direzioni nord, ovest, est e sud, adiacenti a quella centrale). Le celle appartenenti all'intorno vengono etichettate con i seguenti valori: 0, 1, 2, 3, 4 mentre i flussi dalla cella centrale verso le quattro adiacenti sono identificate attraverso i seguenti indici: 0, 1, 2, 3.

Il diagramma sottostante mostra gli indici e le etichette di una cella e dei suoi dintorni:





**S**: è l'insieme degli stati della cella. È suddiviso nei seguenti sottostati:

* : è l'insieme dei valori che rappresentano l'altitudine topografica (es. quota s.l.m.);
* : è l'insieme dei valori che rappresentano lo spessore del fluido;
* sono gli insiemi dei valori che rappresentano i deflussi dalla cella centrale alle quattro celle vicine adiacenti.

**P** = {, }: è l'insieme dei parametri che regolano la dinamica del modello. In particolare, specifica lo spessore minimo al di sotto del quale il fluido non può defluire dalla cellula per effetto dell'aderenza, mentre è il parametro del tasso di rilassamento, un fattore di smorzamento del deflusso.

: → è la funzione di transizione cellulare deterministica. È composto da tre processi elementari, elencati di seguito nell’ordine in cui vengono applicati:

* , imposta i deflussi dalla cella centrale alle celle adiacenti a zero.
* : → , calcola i flussi in uscita dalla cella centrale alle quattro celle adiacenti (nord, sud, ovest, est) applicando l'algoritmo di minimizzazione delle differenze
* : →, determina il valore dello spessore dei detriti all'interno della cella considerando lo scambio di massa nelle vicinanze della cella.

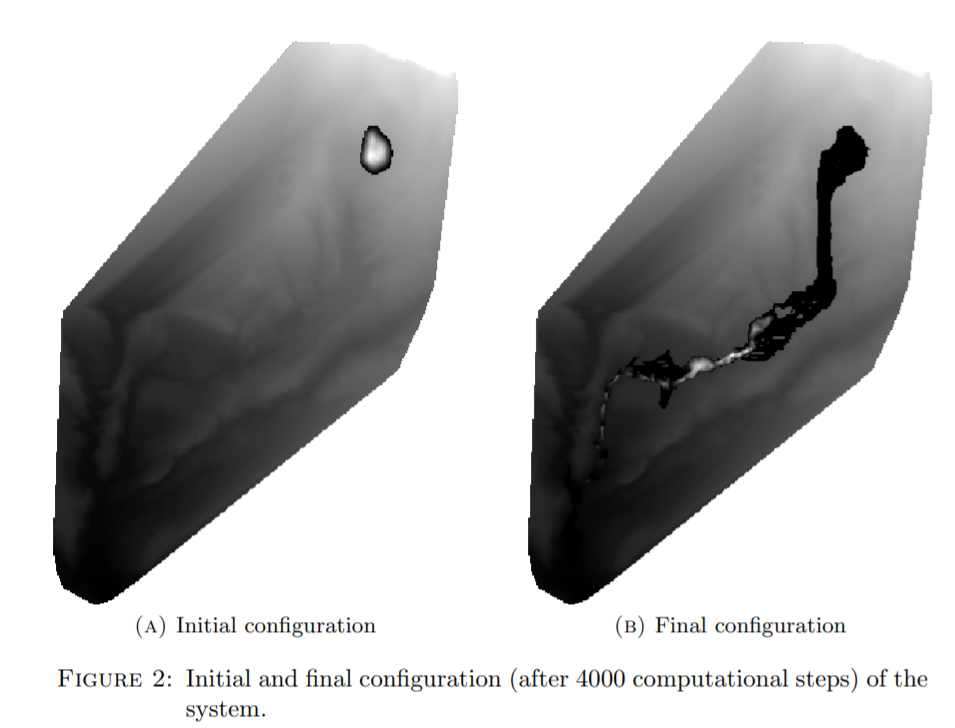
Oltre ai valori da assegnare ai parametri, che sono definiti nel codice sorgente (con = 0.001 e = 0.5), l'input al modello è rappresentato da tre file di testo in formato Ascii. Il primo (l’intestazione) definisce le dimensioni del dominio (numero di righe e colonne), le coordinate geografiche (della cella in basso a sinistra della griglia), la dimensione della cella e un valore no-data (usato per etichettare le celle con mancanza di informazioni). In realtà, vengono prese in considerazione solo le dimensioni del dominio e i valori no-data. I restanti file (mappe a griglia) rappresentano le informazioni riguardanti l'altitudine topografica e lo spessore del flusso per ogni dominio cellula.

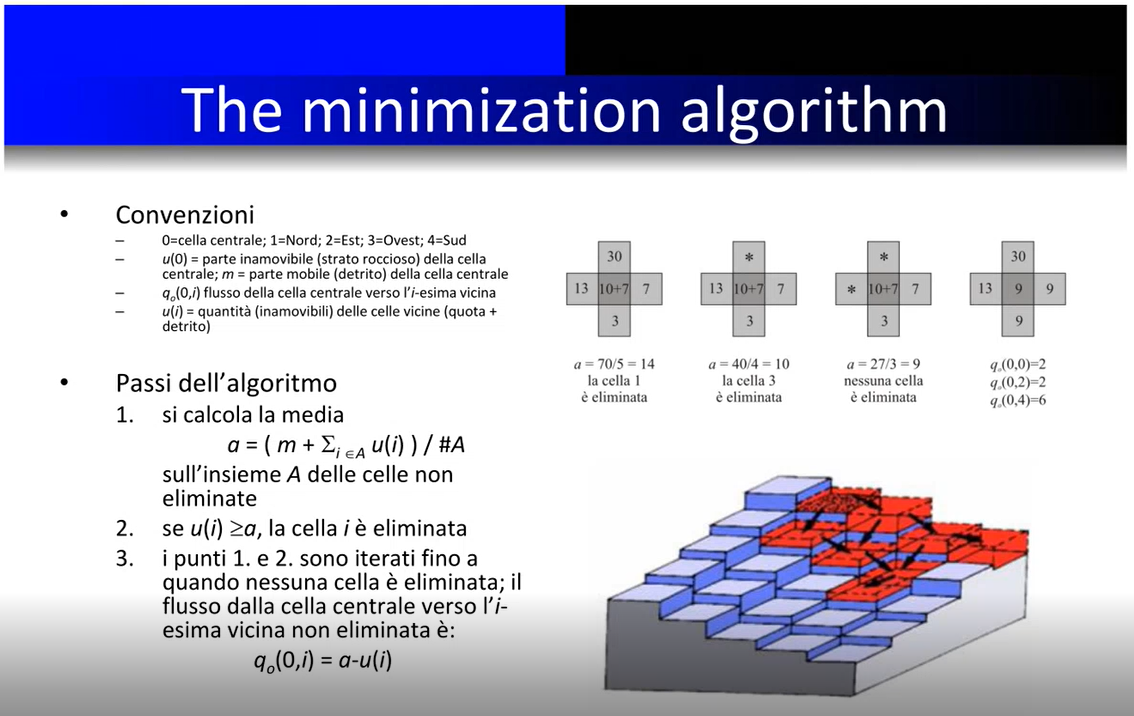
La configurazione iniziale del sistema è definita come segue:

* Una mappa topografica viene letta da un file per inizializzare il sottostato .
* Viene letta una mappa dello spessore di massa da un file per inizializzare il sottostato ;
* I livelli del sottostato di deflusso sono inizializzati a zero ovunque.

Ad ogni iterazione, sono calcolati tre sottopassi di base, corrispondenti ai processi della funzione di transizione sopra descritti.

* Il kernel *sciddicaTResetFlows* corrisponde al processo elementare ;
* Il kernel *sciddicaTFlowsComputation* corrisponde al processo elementare ;
* Il kernel *sciddicaTWidthUpdate* corrisponde al processo elementare .





# Parallel Implementations

## Straightforward parallelization

* *Monolithic*: il kernel non ha modo di gestire il caso in cui il numero di elementi è maggiore del nemero di thread disponibili.
* *Grid-Stride*: risolve il problema del kernel monolithic con i cicli. In particolare, vengono eseguiti più cicli sui dati. Il passo di un ciclo è *blockDim.x\*gridDim.x* che è il numero totale di thread nella griglia. Quindi, se ci sono 1280 thread nella griglia, il thread-0 calcolerà gli elementi 0, 1280, 2560, ecc. Questo è il motivo per cui si chiama Grid-Stride. Usando un ciclo con stride (passo) uguale alla dimensione della griglia, ci assicuriamo la massima coalescenza della memoria, proprio come nella versione monolitica.

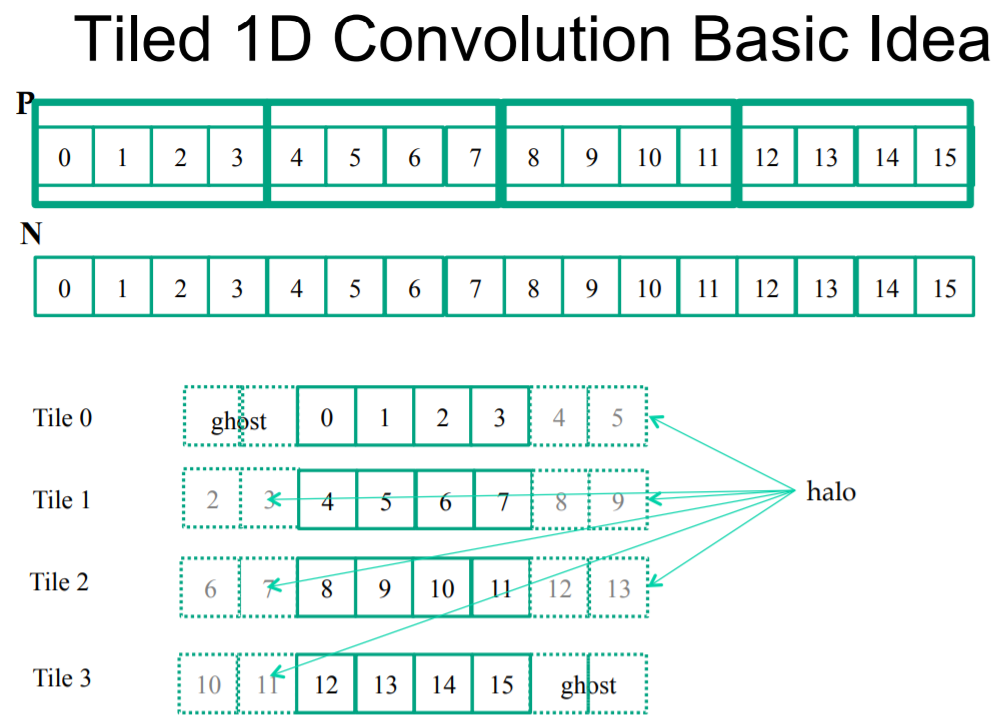
Vantaggi:

* **Scalabilità e riutilizzo dei thread**. Utilizzando un loop, è possibile supportare qualsiasi dimensione del problema anche se supera la dimensione massima della griglia supportata dal dispositivo CUDA. Inoltre, è possibile limitare il numero di blocchi utilizzati per ottimizzare le prestazioni.
* **Debugging**. Usando un loop invece di un kernel monolitico, è possibile passare all'elaborazione seriale avviando un blocco con un thread.
* **Portabilità e leggibilità**. Il codice del Grid-Stride loop è più simile al codice del ciclo sequenziale originale che al codice del kernel monolitico, rendendolo più chiaro per gli altri utenti. In effetti, è possibile scrivere facilmente una versione del kernel che si compila e viene eseguita come kernel CUDA parallelo sulla GPU o come loop sequenziale sulla CPU.

## Tiled parallelization with Halo Cells

Viene fatto uso della tecnica di Tiling, che consiste nel copiare pezzi (o meglio “piastrelle” quadrate) della matrice principale (memoria dichiarata **\_\_global\_\_**) su mini-matrici condivise (memoria dichiarata **\_\_shared\_\_**). In generale, la tecnica Tiling permette di ridurre gli accessi alla memoria globale (più lenta rispetto alla memoria condivisa) e di velocizzare le operazioni. Tutti i valori della matrice che “escono fuori” dalla “piastrella” (tile) vengono poste a 0 e sono chiamate celle Halo. Solo i thread che hanno entrambi gli indici più piccoli della larghezza della tile sono elaborati.

Di seguito, un esempio di Tiled 1D Convolution con Halo cells:



## Tiled parallelization without Halo Cells

Situazione simile alla precedente, ma questa volta non vengono considerate le celle Halo.

## MPI Parallelization

Con la funzione *updateHalo* vengono aggiornate le celle Halo. Nello specifico si sfruttano le funzioni base della libreria MPI, mpi\_Isend e mpi\_Irecv, per consentire la comunicazione NON bloccante tra thread e quindi l’aggiornamento.

Le funzioni *compute\_process* ricevono l'input dal processo server data, computano l'algoritmo ed eventualmente inviano indietro i chunk aggiornati al processo server data.

Prima di tutto, viene allocata sia sull'host che sulla GPU la memoria necessaria per memorizzare il chunk, incluse le halo. Ricevuto il chunk dal server data viene copiato nella GPU. In seguito viene allocata anche la memoria per i dati di output.

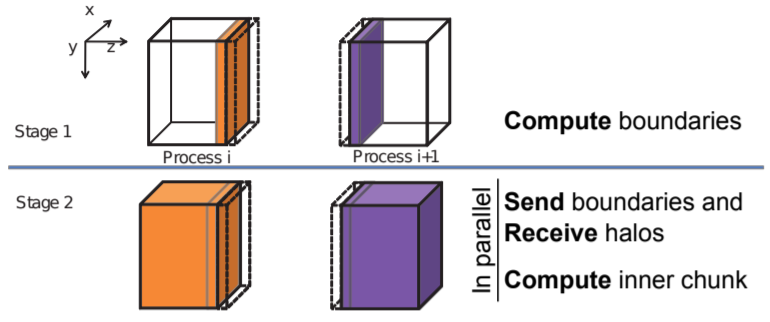
Le attività di calcolo sono suddivise in due fasi:

Fase 1.

* Tutti i processi di calcolo aggiornano le loro celle di confine (sia a sinistra che a destra).

Fase 2.

* Copia delle celle di confine aggiornate nell'host.
* Invio delle celle di confine ai processi adiacenti/ricezione delle celle di confine da processi adiacenti e memorizzazione nelle halo corrispondenti.
* Calcolo della parte interna del blocco.



# Computational Performance

In questa sezione gli algoritmi sono analizzati dal punto di vista delle performance computazionali.

## Monolithic

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | dimBlock(2,2,1)  dimGrid(305,248,1) | dimBlock(4,4,1)  dimGrid(153,124,1) | dimBlock(8,8,1)  dimGrid(76,62,1) | dimBlock(16,16,1)  dimGrid(38,31,1) | dimBlock(32,32,1)  dimGrid(19,16,1) |
| Time [s] | 10.45 | 5.13 | 3.65 | 3.94 | 4.27 |

### **SpeedUp**

67.5/10.45 = 6.46 Tempo di esecuzione seriale = 67.5 s

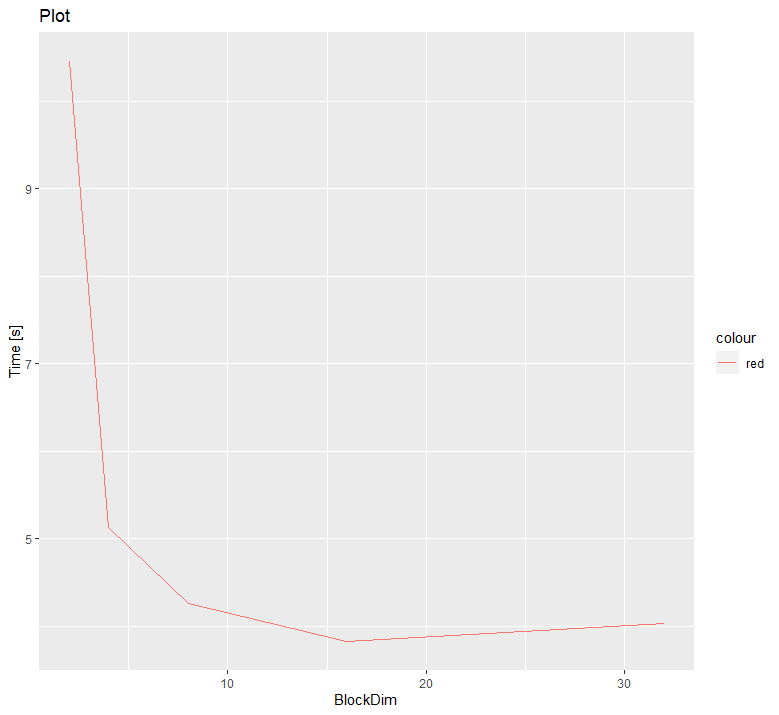
67.5/5.13 = 13.16

67.5/3.65 = 18.49

67.5/3.94 = 17.13

67.5/4.27 = 15.81

### **Plot**



### **Profiling**

dimBlock(8,8,1)

dimGrid(76,62,1)

Immagine che contiene testo

Descrizione generata automaticamente

dimBlock(16,16,1)

dimGrid(38,31,1)

Immagine che contiene testo

Descrizione generata automaticamente

dimBlock(32,32,1)

dimGrid(19,16,1)

Immagine che contiene testo

Descrizione generata automaticamente

## Grid-Stride

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | dimBlock(2,2,1)  dimGrid(305,248,1) | dimBlock(4,4,1)  dimGrid(153,124,1) | dimBlock(8,8,1)  dimGrid(76,62,1) | dimBlock(16,16,1)  dimGrid(38,31,1) | dimBlock(32,32,1)  dimGrid(19,16,1) |
| Time [s] | 10.66 | 5.24 | 3.25 | 3.53 | 4.18 |

### **SpeedUp**

67.5/10.66 = 6.46

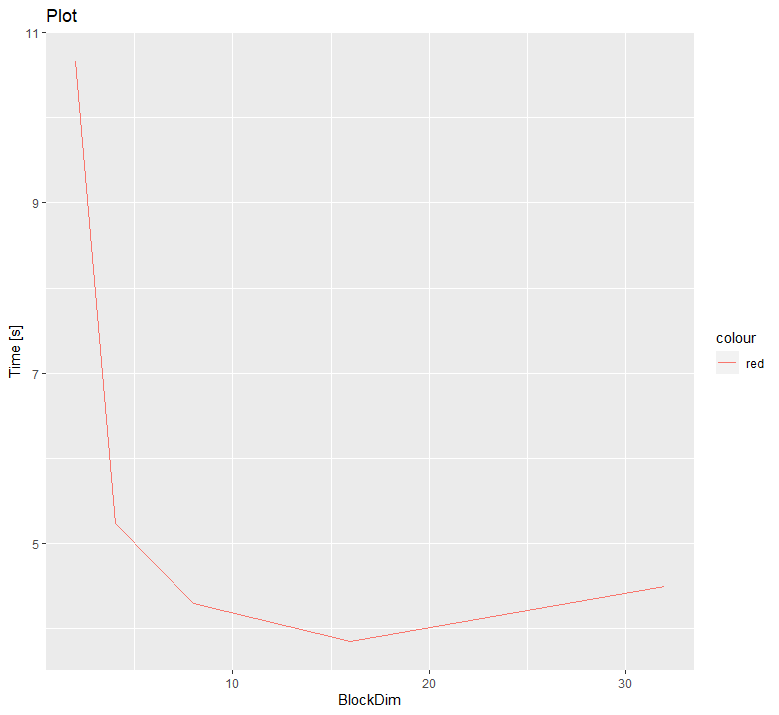
67.5/5.24 = 13.16

67.5/3.25 = 20.77

67.5/3.53 = 19.12

67.5/4.18 = 16.15

### **Plot**



### **Profiling**

dimBlock(8,8,1)

dimGrid(76,62,1)

Immagine che contiene testo

Descrizione generata automaticamente

dimBlock(16,16,1)

dimGrid(38,31,1)

Immagine che contiene testo

Descrizione generata automaticamente

dimBlock(32,32,1)

dimGrid(19,16,1)

Immagine che contiene testo

Descrizione generata automaticamente

## Tiles without halo

|  |  |  |  |
| --- | --- | --- | --- |
|  | tile\_size = 2 | tile\_size = 5 | tile\_size = 10 |
| Time [s] | 12.26 | 3.99 | 3.79 |

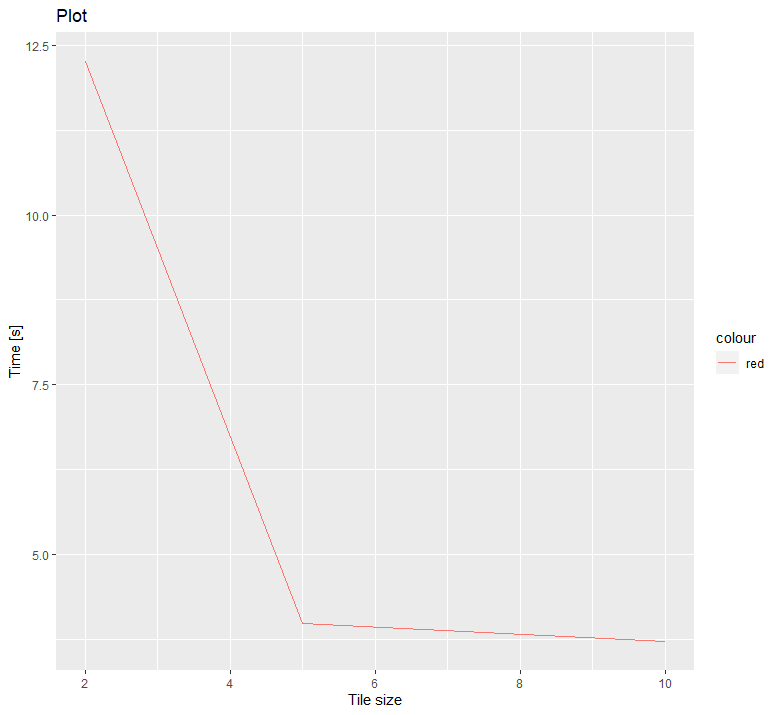
### **SpeedUp**

67.5/12.26 = 5.51

67.5/3.99 = 16.92

67.5/3.79 = 17.81

### **Plot**



### **Profiling**

Tiles\_size = 2

Immagine che contiene testo

Descrizione generata automaticamente

Tiles\_size = 5

Immagine che contiene testo

Descrizione generata automaticamente

Tiles\_size = 10

Immagine che contiene testo

Descrizione generata automaticamente

## Tiles with halo

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | tile\_size = 7  (block\_width=9) | tile\_size = 6  (block\_width=8) | tile\_size = 5  (block\_width=7) | tile\_size = 4  (block\_width=6) | tile\_size = 3  (block\_width=5) | tile\_size = 2  (block\_width=4) |
| Time [s] | 4.10 | 4.44 | 4.66 | 5.51 | 6.32 | 10.58 |

### **SpeedUp**

67.5/4.10 = 16.46

67.5/4.44 = 15.20

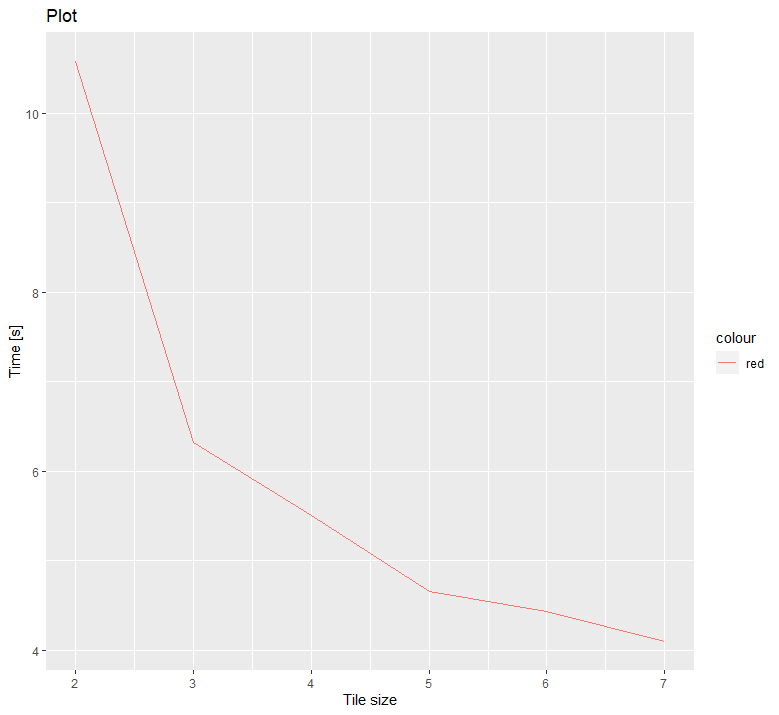
67.5/4.66 = 14.48

67.5/5.51 = 12.25

67.5/6.32 = 10.68

67.5/10.58 = 6.38

### **Plot**



### **Profiling**

mask\_size = 3

tile\_size = 7

block\_width = 9

Immagine che contiene testo

Descrizione generata automaticamente

mask\_size = 3

tile\_size = 6

block\_width = 8

Immagine che contiene testo

Descrizione generata automaticamente

mask\_size = 3

tile\_size = 5

block\_width = 7

Immagine che contiene testo

Descrizione generata automaticamente

# Roofline Assessment

Il modello Roofline è un modello di performance visuale che consente in maniera intuitiva di stimare le performance di un dato kernel computazionale o di una applicazione che esegue su architetture di calcolo di tipo multi-core mostrando graficamente le limitazioni inerenti all'hardware usato e le potenziali ottimizzazioni da poter applicare, nonché la priorità con cui esse necessitano di essere applicate.

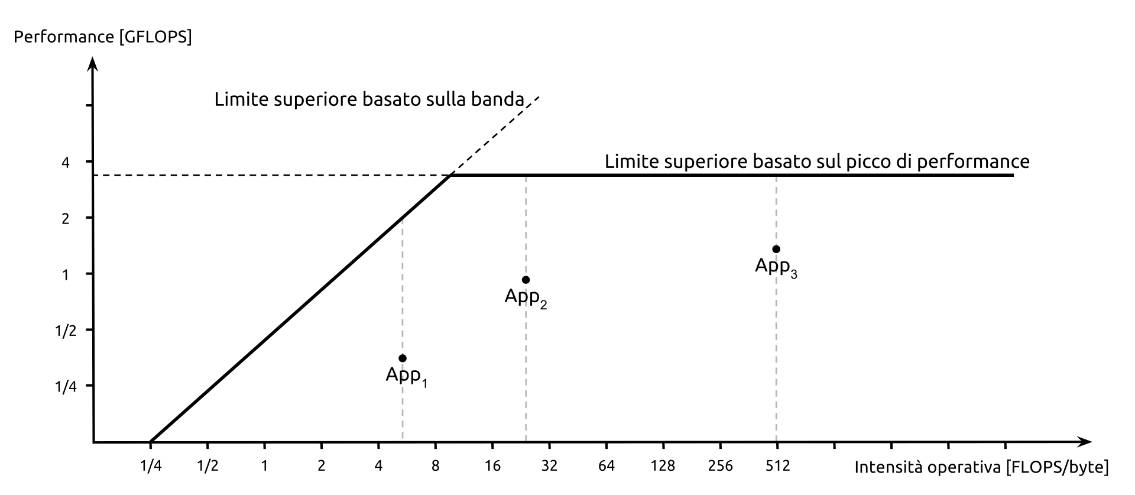
Parametri modello:

* Un parametro tenuto in considerazione dal modello è l’intensità delle comunicazioni, ciò che viene valutato qui è la **larghezza di banda della memoria**, misurata in Gigabyte per secondo (miliardi di byte trasferiti al secondo). Il motivo per cui viene valutato questo aspetto è dovuto al fatto che la memoria ha una grande incidenza sul tempo di risoluzione di un algoritmo poiché le unità aritmetiche devono operare su dati scritti in memoria. Per cui si ricorre a meccanismi di caching per rendere la fase di lettura e scrittura dei dati da e per la memoria più rapidi. Inoltre, anche le ottimizzazioni dal punto di vista software possono aiutare ad evitare dei cache miss quindi a minimizzare le comunicazioni inutili da e verso le memorie.
* Altro aspetto è la **locazione della memoria**: in caso di memoria distribuite più esse saranno numerose, maggiore è la probabilità di un aumento delle comunicazioni; per questo è necessario applicare le tecniche per favorire la località dei dati.

Il modello Roofline consiste in un grafico/tabella che considera le caratteristiche descritte precedentemente, e il fulcro di questo modello è quello di mettere in relazione i GFLOPs con i GByte/s con una quantità chiamata intensità aritmetica (in letteratura anche definita col nome di Computational Intensity). L’intensità aritmetica è la stima del numero di richieste che un software effettua alla memoria, o meglio è il numero di operazioni in virgola mobile per byte di memoria a cui è richiesto l’accesso alla memoria.

Immagine che contiene testo

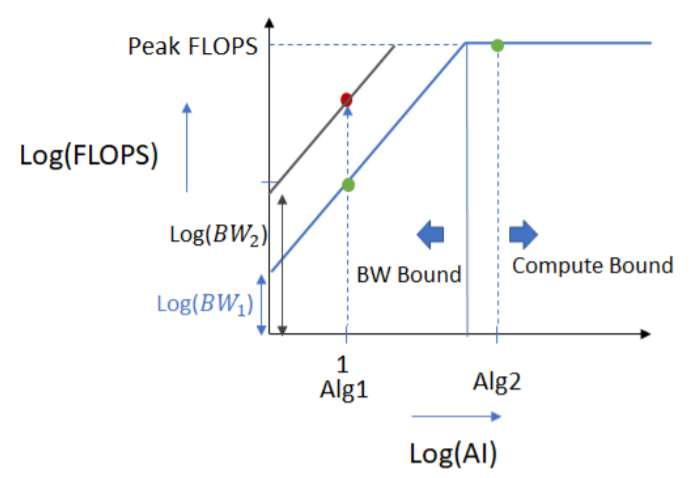
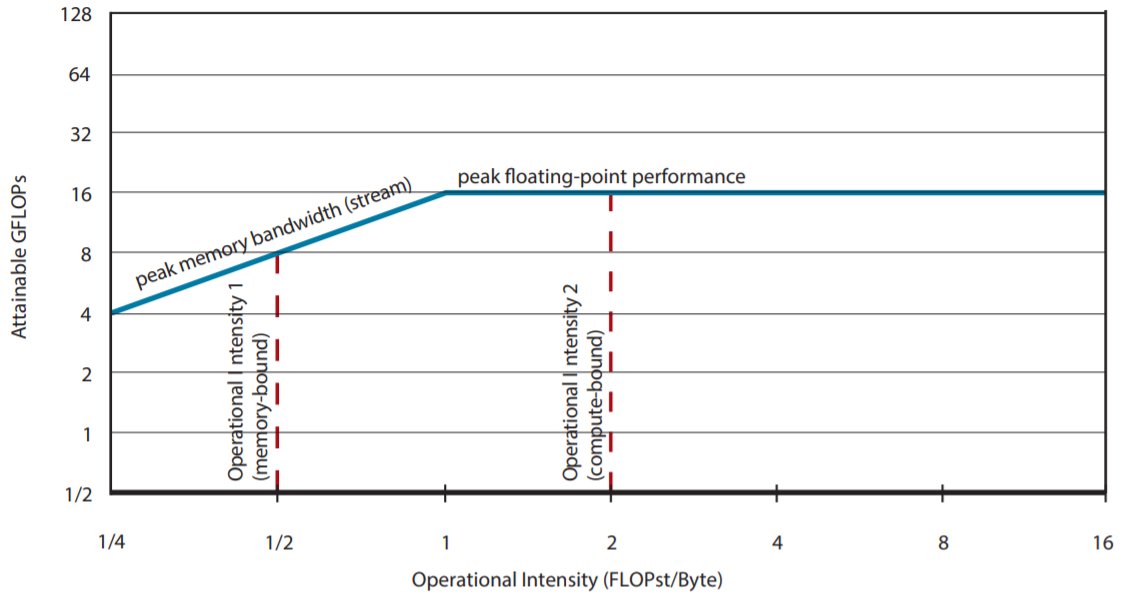
Descrizione generata automaticamente



Se un kernel è **memory-bound**, ovvero quando il tempo affinché ci sia un output dal software è pregiudicato principalmente dalla velocità e dalla capacità della memoria, si ha:

dove è il valore che indica la performance di picco, mentre è il valore che indica il picco del badwidth di memoria.

Mentre un kernel è **compute-bound** si ha che:



### **Specifiche JPDM2**

Theoretical performance: 4612 GFLOP/s (device specs)

Theoretical Global Memory bandwidth: 224.32 GB/s (device specs)

### **Calcolo Intensità aritmetica e applicazione del modello**

Per calcolare l’Intensità aritmetica (AI) è possibile usare la seguente formula:

AI = FP / (TR + TW) \* tempo kernel (dal precedente passo)

FP = floating point operations

TR/TW= dram read/write throughput

## Monolithic

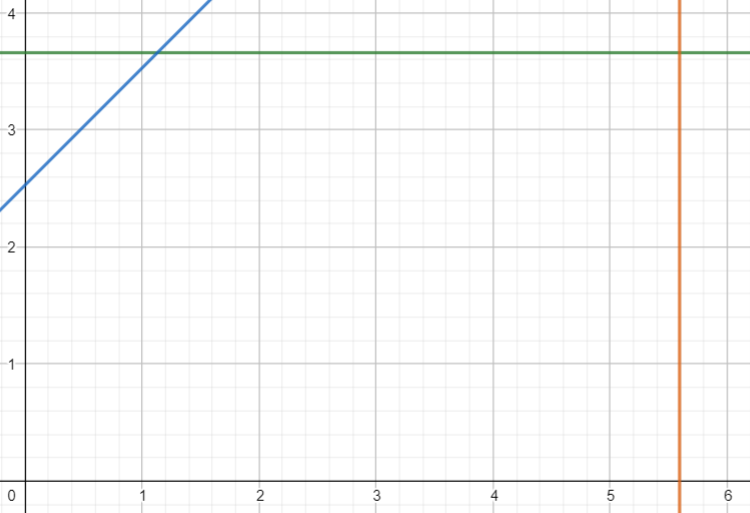
Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo, interni, tastiera, vicino

Descrizione generata automaticamente

AI = 11.472.791/(13,82 + 24,84) \* 1,32 = 391.725 [FLOP/byte]



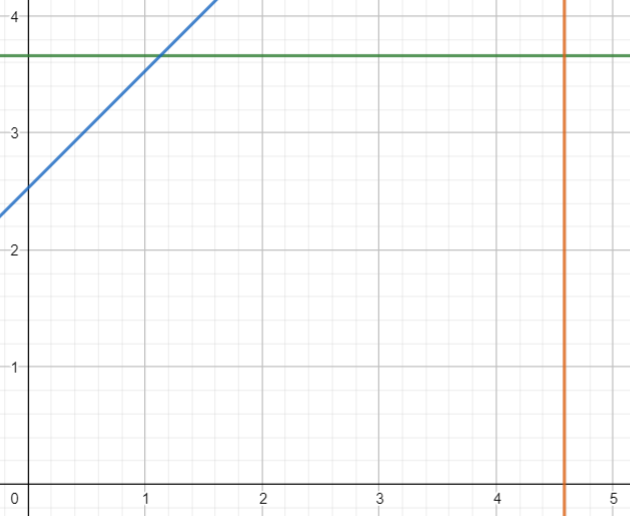
**compute-bound**

Immagine che contiene testo

Descrizione generata automaticamente

955,52 ms = 0,95552 s

AI = 2.402.816/(49,42 + 10,59) \* 0.956 = 38.278 [FLOP/byte]



**compute-bound**

## Grid-Stride

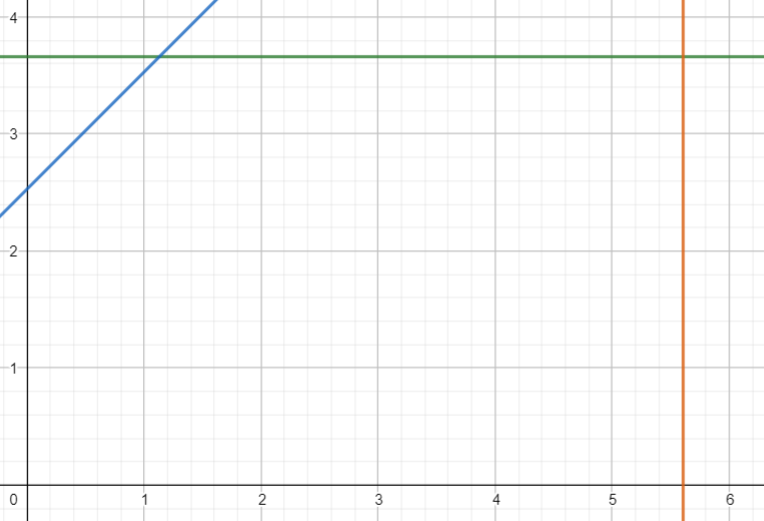
Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

AI = 11.472.791/(13,62 + 24,40) \* 1,34 = 404.354 [FLOP/byte]



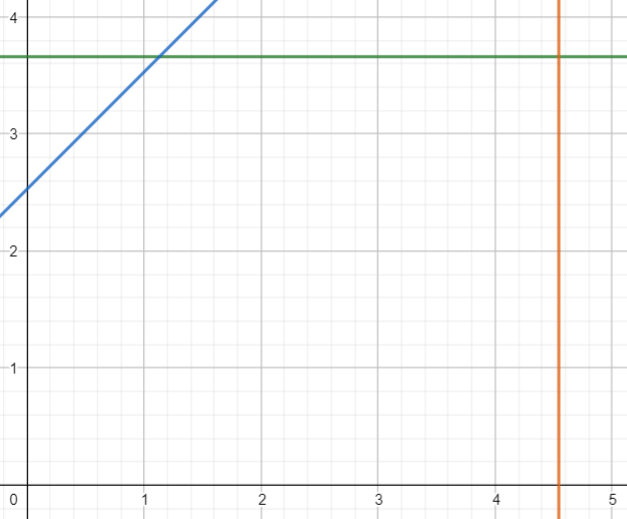
**compute-bound**

Immagine che contiene testo

Descrizione generata automaticamente

914,73 ms = 0,91473 s

AI = 2.402.816/(51,80 + 11,03) \* 0,915 = 34.992 [FLOP/byte]



**compute-bound**

## Tiles without halo

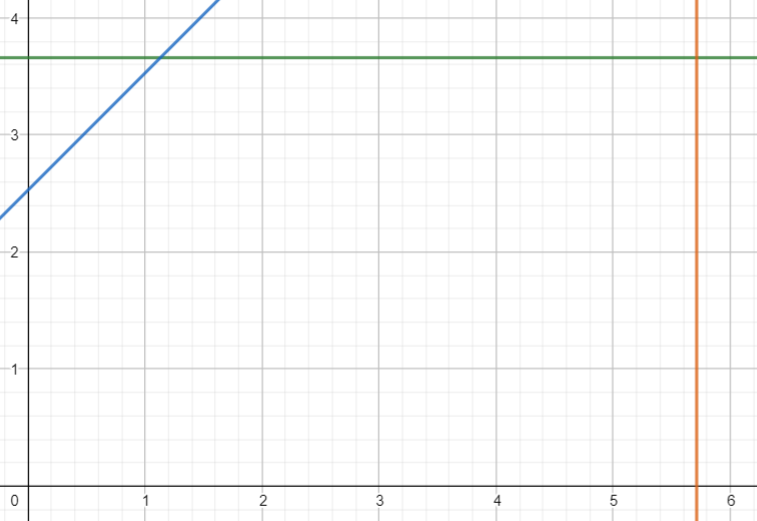
Immagine che contiene testo, computer, tastiera, interni

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

AI = 13.210.859/(12,66 + 23,86) \* 1,43 = 517.293 [FLOP/byte]



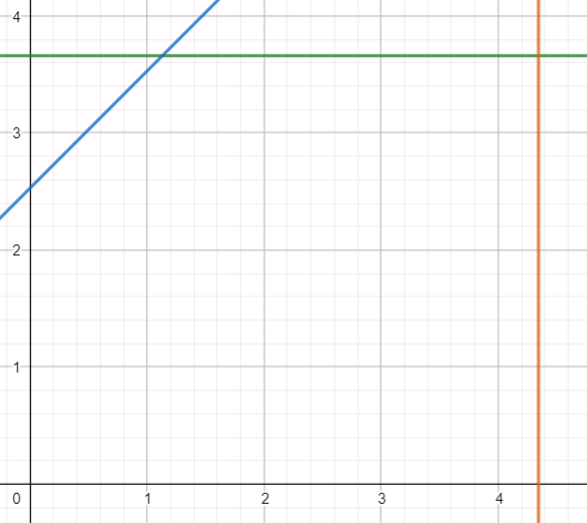
**compute-bound**

Immagine che contiene testo

Descrizione generata automaticamente

697,92 ms = 0,69792 s

AI = 2.402.816/(62,68 + 13,14) \* 0,698 = 22.120 [FLOP/byte]



**compute-bound**

## Tiles with halo

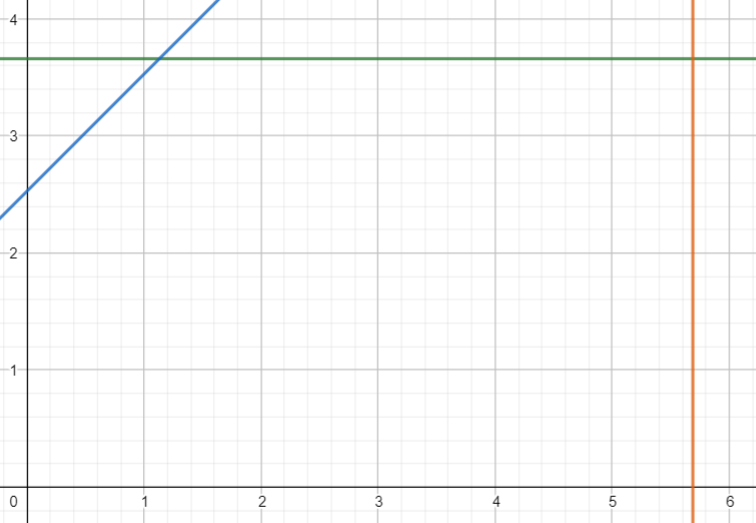
Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo, tastiera, screenshot

Descrizione generata automaticamente

AI = 13.355.597/(20,39 + 32,03) \* 1,92 = 489.179 [FLOP/byte]



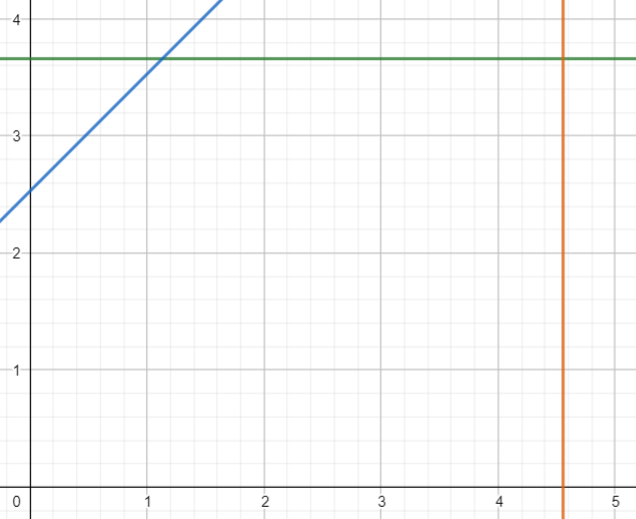
**compute-bound**

Immagine che contiene testo

Descrizione generata automaticamente

884,10 ms = 0,8841 s

AI = 2.394.008/(48,05 + 10,92) \* 0,884 = 35.888 [FLOP/byte]



**compute-bound**

# Conclusions

In questo report sono stati analizzati differenti algoritmi di parallelizzazione di SciddicaT in CUDA. Per raggiungere tale obiettivo ci si è serviti di: strumenti di profilazione di programmi CUDA (come *nvprof*), modelli di performance visuali (come il Roofline Model) e varie misure di performance. La profilazione ha prodotto come output le rispettive statistiche (tempi di esecuzione CPU/GPU, tempi massimi, minimi, ecc.) degli algoritmi e con le opportune opzioni del comando nvprof ha fornito alcune metriche utili alla costruzione del modello Roofline mentre i plot hanno permesso l’analisi anche dal punto di vista grafico. Il Roofline ha fornito informazioni riguardanti la natura dei kernel, e quindi l’appartenenza alla famiglia dei “compute-bound” o dei “memory-bound”.