Progetto di High Performance Computing 2019-2020

Moreno Marzolla moreno.marzolla@unibo.it

Revisioni

- [2020-06-02] Per chi deve ancora consegnare il progetto: **non è necessario includere i file di input** presenti nell'archivio fornito (sono molto voluminosi e problematici da mandare via mail).
- [2019-11-25] Prima versione

Download

- ProgettoHPC1920.zip (ultimo aggiornamento 2019-11-25)
- Versione PDF di questo documento

Descrizione del progetto

Scopo del progetto è la realizzazione di due versioni parallele di un algoritmo per il calcolo dell'*inviluppo convesso* (*convex hull*) di un insieme di punti in due dimensioni. L'inviluppo convesso di un insieme di punti P è il poligono convesso di area minima che li contiene; un esempio è illustrato nella Figura 1.

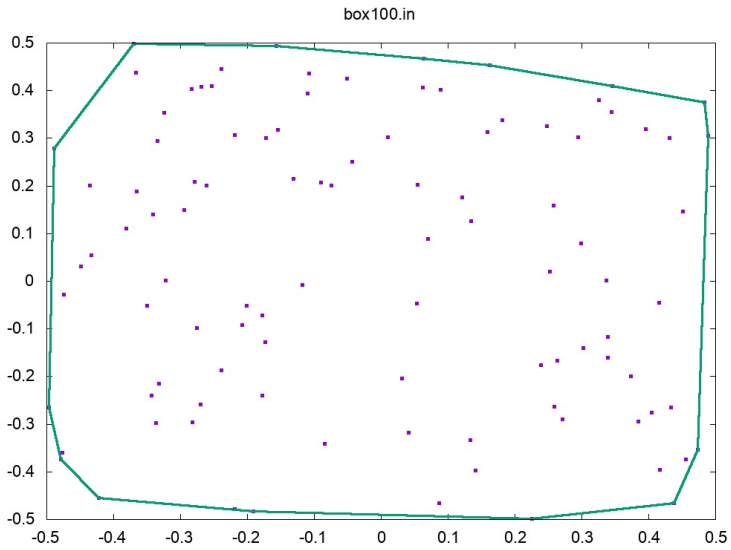


Figura 1: Inviluppo convesso di 100 punti casuali

Il calcolo dell'inviluppo convesso è uno degli algoritmi fondamentali usati in geometria computazionale. Una soluzione efficiente per il calcolo dell'inviluppo convesso è l'algoritmo QuickHull cha è basato su un approccio di tipo *divide et impera*; per calcolare l'inviluppo convesso di n punti, QuickHull richiede tempo $O(n \log n)$ nel caso medio e $O(n^2)$ nel caso pessimo.

Lo scopo di questo progetto è l'implementazione parallela di un algoritmo più semplice ma meno efficiente, noto con il nome *Gift Wrapping* o *Jarvis march*; tale algoritmo determina gli h vertici dell'inviluppo convesso di un insieme di n punti in tempo $O(nh)$. Il nome dell'algoritmo deriva dal suo modo di operare, che è simile all'"incartamento" di un oggetto dalla forma irregolare.

L'algoritmo Gift Wrapping costruisce l'inviluppo convesso in modo incrementale, partendo dal punto più a sinistra (che sicuramente è uno dei vertici dell'inviluppo). Detto `p[cur]` l'ultimo punto aggiunto al poligono, ad ogni passo si aggiunge il punto `p[next]` tale che la spezzata `p[cur]--p[next]--p[j]` sia orientata in senso orario per ogni *j* (immagine a sinistra nella Figura 2).

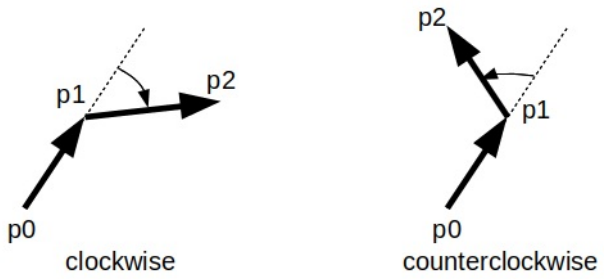


Figura 2: Segmenti che formano un angolo orario (a sinistra) e antiorario (a destra)

L'algoritmo può essere descritto mediante lo pseudocodice seguente.

```
Algorithm gift-wrapping(P[0..n-1]) -> H[0..h-1]
  // P[] is the set of points
  // S[] is the set of corners of the convex hull of P[]
  cur = leftmost = index of leftmost point in P[0..n-1]
  do
    append P[cur] to S[];
    next = (cur + 1) % n;
    for j = 0 to n-1 do
      if (P[cur]--P[next]--P[j] turns left) then
        next = j;
      endif
    endfor
    cur = next;
  while (cur != leftmost)
  return H[]
```

Per illustrare il funzionamento dell'algoritmo consideriamo l'esempio in Figura 3.

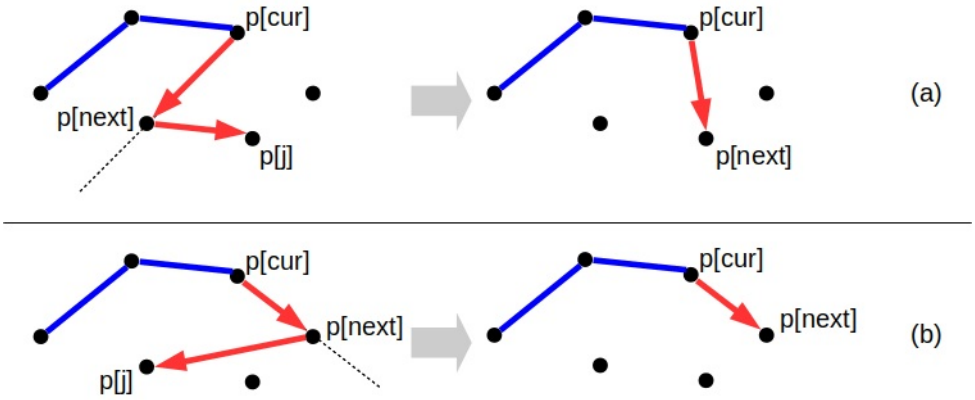


Figura 3: Funzionamento dell'algoritmo Gift Wrapping

Le linee blu rappresentano i lati dell'inviluppo che sono già stati trovati. Indichiamo con `p[cur]` l'ultimo vertice identificato. L'algoritmo esamina tutti i punti `p[j]` per determinare il vertice successivo `p[next]`. Se la spezzata `p[cur]--p[next]--p[j]` forma un angolo antiorario (cioè "curva a sinistra", Figura 3a), allora `p[next]` è un punto interno e non può far parte del bordo dell'inviluppo convesso, mentre `p[j]` potrebbe farne parte. Se invece la spezzata `p[cur]--p[next]--p[j]` forma un angolo orario (cioè "curva a sinistra", Figura 3b), allora `p[j]` è un punto interno e non può far parte dell'inviluppo convesso. Quando tutti i punti sono stati esaminati, `p[next]` diventa il vertice successivo dell'inviluppo convesso, e il procedimento si ripete fino a quando si torna al punto di partenza. Viene fornito un file `convex-hull.c` contenente l'implementazione seriale dell'algoritmo Gift Wrapping.

Non è necessario usare costose operazioni trigonometriche per determinare se una spezzata curva a sinistra o a destra. È però possibile riformulare l'algoritmo in modo da basarsi sull'angolo di curvatura (si veda la Figura 4). L'idea è ancora quella di costruire l'inviluppo convesso in modo incrementale, partendo dal punto più a sinistra e procedendo in senso orario. Chiamiamo `p[prev]` e `p[cur]` gli ultimi due punti aggiunti al poligono. Il punto successivo `p[next]` è quello che minimizza l'angolo orario della spezzata `p[prev]--p[cur]--p[next]` per ogni possibile `p[next]`. Si presenta un problema nel primo passo dell'algoritmo, quando è noto il solo vertice `p[leftmost]` dell'inviluppo. È facile rendersi conto che si può considerare come predecessore "fittizio" un punto virtuale ottenuto traslando `p[leftmost]` verso il basso di una quantità arbitraria.

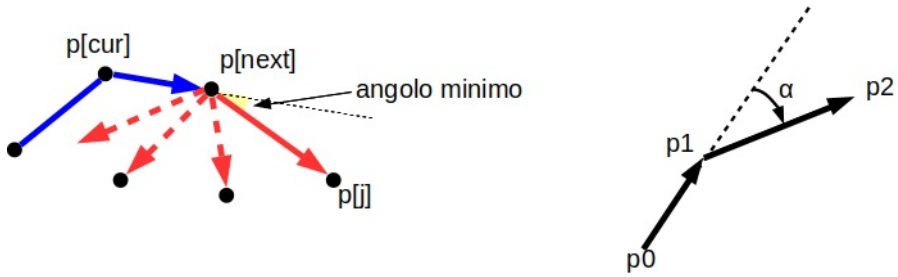


Figura 4: Gift Wrapping con misura esplicita degli angoli

Nel file `convex-hull.c` è presente una funzione `cw_angle(p0, p1, p2)` che calcola l'angolo in senso orario tra tre punti $p0$, $p1$ e $p2$. Tale funzione non viene utilizzata nel programma fornito, ma è lasciata a disposizione di chi la ritenga utile.

Specifiche del progetto

È richiesta la realizzazione di **due** implementazioni parallele dell'algoritmo Gift Wrapping:

- la prima deve essere realizzata usando OpenMP;
- la seconda deve essere realizzata usando uno a scelta tra MPI oppure CUDA.

Come punto di partenza viene fornita una versione seriale `convex-hull.c` che legge da standard input le coordinate di un insieme di punti e stampa a video le coordinate di quelli che fanno parte dell'inviluppo convesso. I dati di input devono rispettare il seguente formato:

- La prima riga contiene il numero di dimensioni dello spazio in cui si trovano i punti (l'unico valore ammesso è 2); gli eventuali caratteri successivi presenti sulla stessa riga vengono ignorati.
- La seconda riga contiene il numero di punti n.
- Le successive n righe contengono le coordinate reali `x` e `y` di ciascun punto, separate da spazi; si usi il tipo `double` per memorizzare le coordinate.

Un esempio di input è il seguente:

```
2 rbox 5 s D2
5
-0.3634838014039983 0.3433358794488259
0.2429460407177192 0.437009406420027
0.2512455848588854 0.4322911704961329
-0.4071324779640412 -0.2902466974607282
-0.3633463862938436 0.3434813001710645
```

Alcuni dei file di input sono stati generati con il comando `rbox` che fa parte del pacchetto `qhull-bin` già installato sul server.

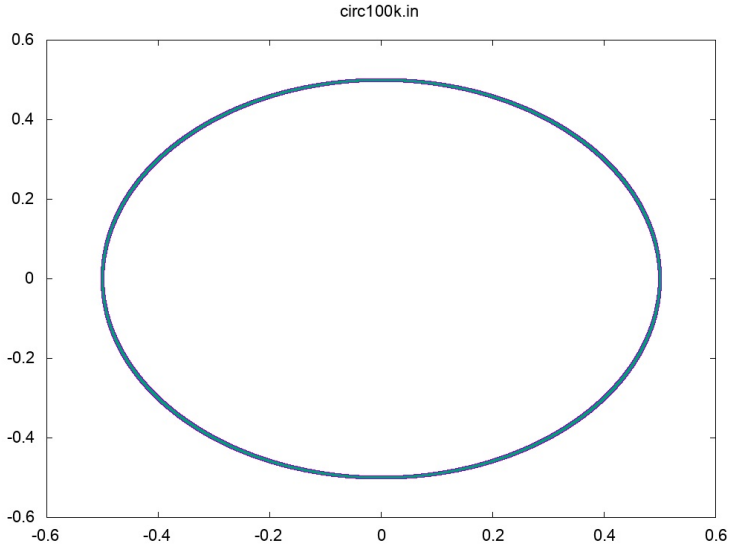
Il programma `convex-hull` stampa su standard output le coordinate dei punti dell'inviluppo convesso, nello stesso formato del file di input; su standard error vengono stampati il numero di vertici dell'inviluppo, il perimetro (*Total facet area*) e l'area (*Total volume*).

> Quando sono presenti punti collineari sul bordo, come nel caso del file `ace.in`, l'inviluppo convesso non è univocamente determinato. Ai fini di questo progetto non occorre calcolare l'inviluppo convesso con il minor numero di vertici. Sarà considerato corretto qualunque implementazione che determini l'inviluppo di area minima, a prescindere dal numero di vertici che lo compongono.

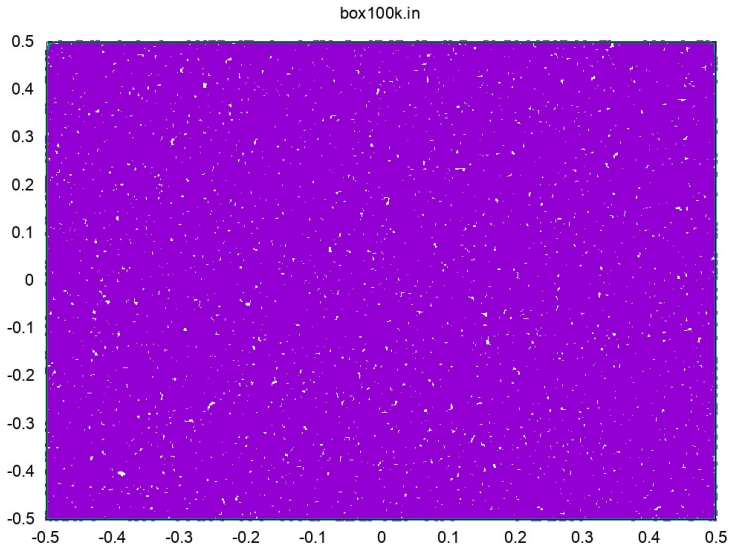
Se l'inviluppo convesso contiene h vertici, il programma stampa le coordinate di $(h + 1)$ punti, in quanto il primo vertice viene ripetuto alla fine. Questo consente di visualizzare graficamente il risultato usando il programma gnuplot assieme allo `plot-hull.gp` fornito. Ad esempio, per visualizzare l'inviluppo convesso dei punti memorizzati nel file `ace.in` si possono usare i comandi seguenti:

```
./convex-hull < ace.in > ace.hull
gnuplot -c "plot-hull.gp" ace.in ace.hull ace.png
```

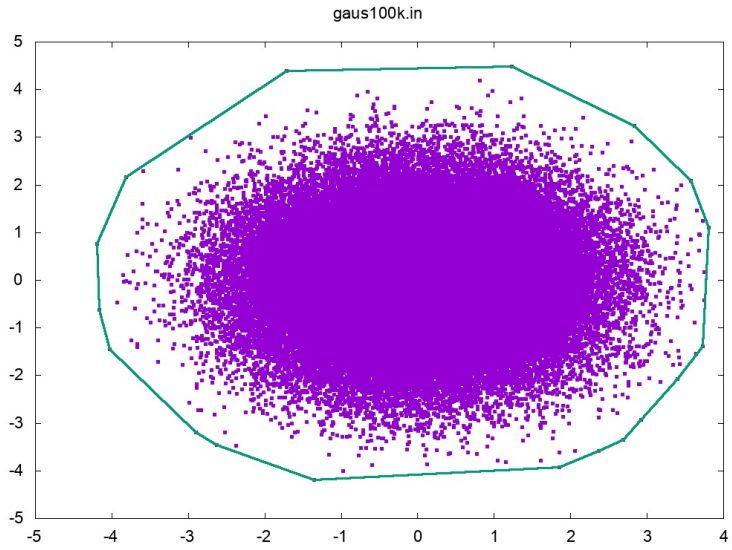
Seguono alcune immagini che mostrano i punti dei file di input (in blu) e l'inviluppo convesso calcolato.



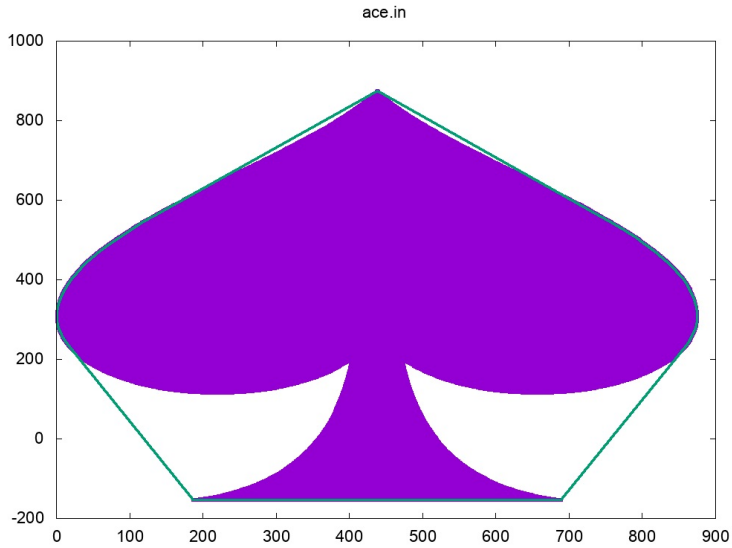
`circ100k.in` (100k punti su una circonferenza)



`box100k.in` (100k punti all'interno di un quadrato)



`gaus100k.in` (100k punti con distribuzione normale)



`ace.in` (485461 punti disposti come l'asso di picche)

Il file `circle100k.in` contiene 100000 punti disposti lungo una circonferenza. Di conseguenza, tutti fanno parte dell'inviluppo convesso e tale file rappresenta il caso pessimo per l'algoritmo Gift Wrapping.

Il formato dei file di input è compatibile con il programma `qconvex`, anch'esso parte del pacchetto `qhull-bin`, che implementa l'algoritmo QuickHull. Per calcolare l'inviluppo convesso e visualizzare alcune informazioni su di esso è possibile dare il comando

```
qconvex FA < ace.in
```

(i flag `FA` servono per mostrare l'area dell'inviluppo) che produce un output simile al seguente

```
Convex hull of 485461 points in 2-d:

  Number of vertices: 66
  Number of facets: 66

Statistics for:  | qconvex

  Number of points processed: 66
  Number of hyperplanes created: 130
  Number of distance tests for qhull: 3308986
  CPU seconds to compute hull (after input): 0.04027
  Total facet area:   2960.9744
  Total volume:       617825.5
```

Si noti che, nel caso bidimensionale, *Total facet area* indica il perimetro, mentre *Total volume* indica l'area dell'inviluppo convesso.

L'algoritmo QuickHull è mediamente più efficiente di Gift Wrapping, anche in versione parallela, per cui le prestazioni non sono confrontabili. Inoltre, a differenza del programma fornito, il comando `qconvex` sembra calcolare sempre l'inviluppo con il numero minimo di vertici; per confrontare il risultato con il proprio programma occorre quindi fare riferimento al valore dell'area (*Total volume*),

Limiti

Le versioni parallele consegnate devono funzionare correttamente (almeno) su tutti i file di input allegati al progetto; naturalmente verranno testate anche su input diversi. Si può assumere che il numero di punti n sia sempre minore o uguale a $1024^2 = 1048576$. Relativamente alle sole versioni OpenMP e MPI, si può inoltre assumere che i programmi verranno sempre testati con un numero di punti molto maggiore del numero di processi/thread impiegati.

Programmi che impongono un vincolo sul numero di punti (esempio, n deve essere multiplo del numero di processi MPI o di thread OpenMP) verranno accettati ma otterranno una valutazione inferiore di programmi corretti che funzionano per qualsiasi valore di n (soggetto ai limiti di cui sopra).

Cosa consegnare

È richiesta la consegna di **due** versioni parallele del programma:

1. La prima, chiamata `omp-convex-hull.c`, deve utilizzare OpenMP;
2. La seconda deve essere basata, a scelta, su MPI oppure CUDA (uno dei due, non entrambi). Il file deve chiamarsi rispettivamente `mpi-convex-hull.c` oppure `cuda-convex-hull.cu`

Oltre ai due programmi di cui sopra, è obbligatorio includere:

3. Una relazione in formato PDF della lunghezza massima di 6 facciate che descriva le strategie di parallelismo adottate e discuta scalabilità ed efficienza dei programmi realizzati.

Ulteriori requisiti:

- Includere nome, cognome e numero di matricola in un commento all'inizio di tutti i file sorgenti consegnati e nella relazione.
- Il codice sorgente deve essere comprensibile e adeguatamente commentato.
- Includere un file `README` contenente le istruzioni per la compilazione e l'esecuzione dei programmi consegnati; chi lo desidera può usare un `Makefile` per la compilazione (consigliato; ci si può basare su quello fornito).
- Consegnare tutti i file necessari per la compilazione, incluso l'header `hpc.h` visto a lezione (per chi lo usa).
- I programmi verranno compilati e testati sul server `isi-raptor03.csr.unibo.it`. I programmi che non compilano o non eseguono correttamente su tale macchina non saranno presi in considerazione.

- La relazione non deve superare la lunghezza di **sei facciate** in formato A4, contando tutte le pagine (inclusi eventuali frontespizi, appendici e quant'altro). Il formato della relazione è libero; a titolo puramente indicativo viene fornito uno schema in formato LibreOffice.
- La relazione non deve commentare il codice riga per riga (per quello ci sono già i sorgenti), ma deve illustrare le strategie di parallelizzazione adottate, discutere eventuali vantaggi e limitazioni delle scelte effettuate, e mostrare la scalabilità dei programmi mostrando i grafici di *speedup* ed *efficienza* dove abbiano senso.

Come consegnare

Il progetto può essere consegnato in qualsiasi momento **entro le ore 23:59 del 30 settembre 2020**. La consegna deve avvenire inviando una mail dal proprio indirizzo istituzionale `@studio.unibo.it` a moreno.marzolla@unibo.it con oggetto:

> [HPC] Consegna Progetto 2019/2020

Nella mail vanno indicati cognome, nome, numero di matricola del mittente. Alla mail deve essere allegato un archivio in formato `.zip` oppure `.tar.gz` contenente i sorgenti e la relazione. L'archivio sarà denominato con il cognome e nome dell'autore (es., `MarzollaMoreno.zip` oppure `MarzollaMoreno.tar.gz`), e dovrà contenere una directory con lo stesso nome (es., `MarzollaMoreno/`) con il seguente layout:

> `MarzollaMoreno/src/omp-convex-hull.c`
>
> `MarzollaMoreno/src/mpi-convex-hull.c` (se si svolge la versione MPI)
>
> `MarzollaMoreno/src/cuda-convex-hull.cu` (se si svolge la versione CUDA)
>
> `MarzollaMoreno/src/...` (ogni altro file necessario alla compilazione e/o esecuzione del codice)
>
> `MarzollaMoreno/src/Makefile` (facoltativo)
>
> `MarzollaMoreno/README`
>
> `MarzollaMoreno/Relazione.pdf`
>
> (altri file se necessario)

Riceverete una mail di conferma dell'avvenuta ricezione del progetto; l'invio di tale mail potrebbe richiedere alcuni giorni, per cui si prega di pazientare.

È possibile consegnare il progetto prima o dopo aver sostenuto la prova scritta. Le eventuali valutazioni positive (del progetto e/o della prova scritta) **restano valide fino al 30 settembre 2020**; dopo tale data inizierà la nuova edizione del corso e **tutti i voti in sospeso verranno persi**.

Il progetto si consegna una volta sola. Non è possibile apportare modifiche o correzioni dopo la consegna: chi vuole migliorare la valutazione dovrà consegnare un nuovo progetto su nuove specifiche fornite dal docente.

Come svolgere il progetto

- *Il progetto deve essere svolto individualmente*. Non è consentito condividere in tutto o in parte il codice o la relazione con altri studenti.
- *Il codice del progetto deve essere sviluppato in autonomia*. È consentito l'uso di porzioni di codice reperito in rete o tramite altre fonti purché (i) la provenienza di ogni frammento di codice scritto da terzi sia chiaramente indicata in un commento, e (ii) la licenza di tale codice ne

consenta il riutilizzo. Come unica eccezione, il codice reso disponibile dal docente durante le lezioni o i laboratori può essere usato liberamente senza necessità di indicarne la fonte.
- Sono disponibile a fornire chiarimenti sulle specifiche del progetto (cioè sul presente testo), ma **non guarderò il vostro codice per nessun motivo se non dopo la consegna**. Lo svolgimento in autonomia del progetto è un requisito dell'esame e va preso con la massima serietà.

Valutazione del progetto

Sebbene il progetto possa essere consegnato in qualsiasi momento, effettuerò tre sessioni di valutazione al termine delle sessioni d'esami di gennaio/febbraio 2020, giugno/luglio 2020 e settembre 2020. Questo significa che alla fine di febbraio 2020, luglio 2020 e settembre 2020 valuterò tutti i progetti ricevuti fino a quel momento. Chi avesse delle scadenze, ad esempio legate a richieste di borse di studio o per potersi laureare, è pregato di segnalarmelo all'atto della consegna. L'unico vincolo è che il progetto venga consegnato **almeno 10 giorni lavorativi prima della data entro la quale si richiede la correzione** (sottolineo **lavorativi**) per consentirmi di far fronte anche agli altri impegni accademici.

Un progetto verrà considerato sufficiente se soddisfa almeno i seguenti requisiti minimi:

- Il progetto compila ed esegue correttamente su istanze di input anche soggette a vincoli (es., numero di punti multiplo di…) sul server `isi-raptor03.csr.unibo.it`.
- La relazione dimostra un livello sufficiente di padronanza degli argomenti trattati.

Ulteriori aspetti che potranno comportare una valutazione superiore:

- Qualità della relazione, in termini di correttezza e chiarezza.
- Qualità del codice, in termini di chiarezza e uso appropriato delle primitive e/o dei pattern di programmazione concorrente adeguati.
- Generalità della soluzione (es., la soluzione proposta funziona per qualunque dimensione del dominio, oppure con qualunque numero di processi MPI, ecc.).

Il voto del progetto sarà espresso in trentesimi (massimo 30); si terrà conto di progetti di qualità particolarmente elevata per arrotondare in modo favorevole il voto finale, o per assegnare la lode.

Suggerimenti

Considerazioni generali

La versione seriale deve essere considerata come un ausilio per capire i dettagli dell'algoritmo Gift Wrapping. L'implementazione fornita può essere modificata a piacere per adattarla alle proprie esigenze, ad esempio modificando l'algoritmo, o memorizzando i punti in due array `x[]` e `y[]` di coordinate, anziché in un unico array di strutture `point_t` (abbiamo parlato a lezione di *Array di Strutture* vs *Strutture di Array*).

Va parallelizzata la funzione `convex-hull()`; non è richiesta la parallelizzazione delle altri funzioni accessorie fornite.

Non cercare l'efficienza a tutti i costi. Non è banale parallelizzare in modo efficiente l'algoritmo Gift Wrapping, nemmeno nella versione OpenMP. Esistono diversi livelli di compromesso tra complessità del codice ed efficienza. Per ottenere un buon voto non è richiesta la massima efficienza (che potrebbe non essere nemmeno raggiungibile), ma è necessario compiere scelte oculate. I pro e contro delle soluzioni proposte possono essere discussi nella relazione.

Versione OpenMP

La versione OpenMP può essere ottenuta partendo dalla versione seriale, anche se probabilmente sarà richiesto un pò di lavoro dato che non basta aggiungere un `#pragma omp parallel for`. Consiglio di riflettere bene sulla versione OpenMP, dato che potrebbe costituire un buon punto di

partenza per le versioni MPI o CUDA.

Versione MPI

La versione MPI è laboriosa ma non dovrebbe presentare problemi particolari. Alcune osservazioni specifiche:

- Nell'implementazione dello standard MPI fornita dal pacchetto OpenMPI (installato sul server), l'unico processo che ha accesso a `stdin` è il processo 0; tutti gli altri hanno `stdin` redirezionato a `/dev/null`. Pertanto, il processo 0 è l'unico che può leggere l'input, e sarà quindi responsabile per comunicarlo (in tutto o in parte) agli altri processi.
- Chi lo ritiene utile può definire un nuovo tipo `MPI_point_t` che rappresenti una coppia di valori di tipo `double` utilizzando la funzione `MPI_Type_contiguous()` vista a lezione:

```
MPI_Type_contiguous(2, MPI_DOUBLE, &MPI_point_t);
MPI_Type_commit(&MPI_point_t);
```

- Nel caso fosse necessario effettuare una riduzione per ottenere sia il valore minimo di un insieme di dati sia un indice ad esso associato, si può utilizzare `MPI_Reduction` usando l'operatore `MPI_MINLOC` (oppure `MPI_MAXLOC` per il massimo), come illustrato a lezione. Si faccia riferimento alla man page di `MPI_Reduction` per i dettagli.

Versione CUDA

La versione CUDA presenta più o meno lo stesso livello di difficoltà della versione MPI, ma non è semplice renderla più efficiente della versione OpenMP (è abbastanza problematico renderla più efficiente persino della versione seriale!). Suggerisco di iniziare con una versione almeno funzionante, e cercare di migliorarla fin dove possibile. Ripeto che non è necessario realizzare le versioni parallele più efficienti in assoluto per avere una buona (ottima) valutazione.

Consigli sulla stesura della relazione

Invito a prestare la massima attenzione alla stesura della relazione. Una relazione di qualità scadente è uno dei motivi più frequenti di penalizzazione della valutazione dei progetti. Vanno assolutamente evitati errori grossolani come ~~"ghost shell"~~ invece di "ghost cell", ~~"pull di thread"~~ invece di "pool di thread" (sì, ho visto questo, e di peggio).

Anche se la relazione non è una tesi di laurea, invito a prendere visione dei consigli per la stesura della tesi sulla mia pagina Web.

Checklist per la consegna

Assicurarsi che i seguenti requisiti minimali siano soddisfatti prima di consegnare il progetto:

- Vengono consegnate due versioni del programma, una basata su OpenMP, e una basata (a scelta) su MPI oppure CUDA.
- I sorgenti compilano ed eseguono correttamente sul server `isi-raptor03.csr.unibo.it`.
- La relazione è in formato PDF e ha lunghezza minore o uguale a sei facciate.
- I sorgenti e la relazione indicano chiaramente cognome, nome e numero di matricola dell'autore/autrice.
- Il progetto viene consegnato in un unico archivio in formato `.zip` oppure `.tar.gz`, nominato con cognome e nome dell'autore/autrice (es., `MarzollaMoreno.zip` oppure `MarzollaMoreno.tar.gz`), che include i sorgenti e la relazione in formato PDF.
- La mail di consegna ha oggetto "[HPC] Consegna progetto 2019/2020"; la mail viene spedita dal proprio indirizzo di posta istituzionale, e include cognome, nome e numero di matricola.

Riferimenti bibliografici

- Jarvis, R. A. (1973). "On the identification of the convex hull of a finite set of points in the plane".

Information Processing Letters. 2: 18–21. doi:10.1016/0020-0190(73)90020-3

- La voce di Wikipedia che tratta gli algoritmi per l'inviluppo convesso è un buon punto di partenza per approfondimenti, soprattutto i riferimenti bibliografici.