UNIVERSITÀ DI PISA

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

CORSO DI LAUREA SPECIALISTICA IN TECNOLOGIE INFORMATICHE

TESI DI LAUREA

Studio sulle proprietà di assorbimento energetico degli algoritmi

CANDIDATO

Davide Morelli

RELATORE

Dr. Antonio Cisternino

CONTRORELATORE

Prof. Fabrizio Luccio

CORELATORI

Prof. Paolo Ferragina

Dr. Massimo Coppola

ANNO ACCADEMICO 2009/2010

Ringraziamenti

Molte sono le persone che devo ringraziare per il traguardo che ho raggiunto: il mio relatore Dr. Antonio Cisternino per avermi costantemente motivato, per avermi formato e per aver cambiato completamente la mia visione dell'Informatica, per avermi offerto opportunità a cui altrimenti non avrei mai potuto avere accesso ma soprattutto per la sua amicizia; i miei co-relatori Prof. Paolo Ferragina e Dr. Massimo Coppola, che insieme ad Antonio con pazienza infinita hanno preso i miei appunti ed i risultati dei miei esperimenti e li hanno trasformati in un lavoro organico; Maurizio Davini per l'amicizia e le occasioni offerte, tra cui l'opportunità di partecipare all'International Green Computing Conference di Chicago durante il quale mi sono potuto confrontare con la comunità che si occupa di consumo energetico del software; il Dr. Daniele Mazzei ed il Dott. Carmelo De Maria hanno dato un contributo essenziale al lavoro descrivendomi l'approccio seguito negli esperimenti in coltura cellulare; Cristian, Gabriele, Stefano, Nicole e gli altri amici del VSLab per i consigli con cui mi hanno aiutato con i vari dubbi emersi durante la stesura della tesi; il mio socio Luca per l'amicizia e la pazienza che ha dimostrato durante le mie frequenti assenze dal lavoro; tutti i miei amici per i momenti di gioia che mi hanno regalato; i miei genitori, i miei nonni, mia sorella ed i miei nipoti per aver creduto in me e per avermi sostenuto a tutti i livelli in questi anni di studio; la famiglia di mia moglie per avermi accolto con affetto; Bianca per l'amore con cui ogni giorno condivide con me la sua vita.

Introduzione

I programmi sono costituiti da istruzioni che manipolano le risorse di sistema per raggiungere un obiettivo. Queste istruzioni sono eseguite da moduli di elaborazione che usano segnali elettrici per modificare la configurazione dei bit in base alla semantica dell'istruzione eseguita. Essendo gli attuali calcolatori basati su reti logiche non conservative [1] la manipolazione dei bit nel sistema dissipa necessariamente una certa quantità di energia. Un algoritmo quindi, per essere eseguito, necessita di una quantità di energia legata alla sua complessità computazionale. Ci chiediamo quindi entro quali limiti si riesca a costruire una correlazione tra computazione astratta ed assorbimento energetico e se sia possibile ed utile un'analisi energetica degli algoritmi.

La prima domanda che ci siamo posti è stata quindi se sia misurabile (e con quale accuratezza) l'incremento del consumo energetico di un algoritmo all'aumentare della dimensione di input, analizzando in prima istanza i sistemi di misurazione energetica proposti in letteratura. La quasi totalità dei lavori trovati però si propone di misurare le performance energetiche dell'hardware, senza chiedersi quale relazione ci sia tra la complessità computazione in tempo ed il consumo energetico del software. Sono stati generalmente analizzati sistemi embedded, nei quali le unità di computazione sono più semplici che nei general purpose processor, spesso basati su set di istruzioni RISC e mancanti di molte delle funzionalità che richiedono un alto consumo energetico (multimedia, operazioni in virgola mobile) che sono invece alla base di qualsiasi personal computer. Non stupisce quindi che molti autori siano giunti alla conclusione che in tali sistemi le istruzioni abbiano un costo energetico simile e che abbiano di conseguenza creato modelli di costo e di comportamento regolari [2]. Ma le moderne CPU sono molto più sofisticate di questo modello, in termini di complessità e di numero di transistor, e sono intrinsecamente parallele e concorrenti a livello micro-architetturale. In questo contesto la quantità di energia richiesta per eseguire una singola istruzione è stimata come media nel migliore dei casi. Tali costi possono variare largamente e la loro differenza cresce come si risale lo stack sofware. Si prendano come esempio le macchine virtuali: qui possiamo solo associare un consumo medio a complesse entità come gli opcode del linguaggio intermedio, perché essi possono

scatenare effetti collaterali (energeticamente) costosi. Diventa ancora più complicato definire modelli di costo per programmi che prevedono anche la compilazione. Questo è il motivo per cui sono stati chiesti *"models to be developed at all abstraction levels and granularities. […] These models will provide a solid baseline for higher level models, and many intermediate levels that can be of use to various layers in the abstraction hierarchy."* [3].

In letteratura abbiamo individuato due approcci principali alla modellazione del consumo energetico dei sistemi:

- Accurati modelli analitici basati su misurazioni esplicite dell'energia consumata da insiemi di benchmark a livello hardware [4] [5]. Questo approccio è stato sperimentato solo su ristrette classi di processori semplici le cui caratteristiche di consumo energetico sono conosciute e definite, quali sistemi embedded e micro controllori.
- Modelli basati su simulazione [6] [7]. In questo caso il consumo energetico della macchina virtuale sottostante [8] o dell'intero sistema operativo sono presi in considerazione. Questo approccio è stato applicato per effettuare la profilazione di codice semplice.

Abbiamo tuttavia scelto di implementare un nuovo sistema di misurazione, trovando quelli fino ad ora proposti eccessivamente complessi, caratteristica che avrebbe reso gli esperimenti difficili da replicare ed i dati complessi da interpretare.

Certamente il consumo energetico dei programmi dipende fortemente dal numero di istruzioni eseguite e dal numero di accessi nella gerarchia di memoria, questi sono gli stesso fattori che determinano il tempo di completamento [9]. Però, come è stato chiarificato dal dibattito RISC/CISC di qualche anno fa [10], istruzioni semplici comportano un maggior numero di istruzioni per lo stesso algoritmo, sebbene queste richiedano un ciclo di clock più breve. Non sorprende quindi che [9] sostenga che *"the correspondence between completion time and energy consumption is not one-to-one. […] The average power consumption and computation rates are intricately tied together, making it difficult to speak of power complexity in isolation. […] This also indicates that models for the study of energy-computation tradeoffs would need to address more than just the CPU."*. Ma in letteratura manca un modello astratto, simile

al Modello ad Accesso Casuale (RAM) [11], ed una metodologia, smile all'analisi asintotica della complessità computazionale, che prenda in considerazione le problematiche energetiche nel design e nell'implementazione degli algoritmi.

In questo lavoro proponiamo due contributi a queste problematiche presentando una *teoria* e una *metodologia* per la valutazione energetica degli algoritmi, importante per permettere un design energicamente consapevole dei programmi. La proposta si articola attraverso i seguenti passi:

- Nel secondo capitolo analizzeremo la letteratura alla ricerca di metodologie già esistenti e metteremo in evidenza le caratteristiche principali dei vari filoni
- Nella prima parte del terzo capitolo introdurremo una *metodologia sperimentale* capace di misurare il comportamento energetico dei programmi in modo semplice ma espressivo, questa metodologia permette di confrontare diverse architetture, configurazioni e algoritmi, da chiunque e senza che siano richieste particolari abilità ingegneristiche o di programmazione. La nostra metodologia consiste di alcuni manufatti di semplice realizzazione e dell'uso di software open source. Il tutto sarà descritto nel dettaglio in modo da rendere il processo e gli esperimenti riproducibili (elemento fondamentale di ogni esperimento scientifico).
- Nella seconda parte del terzo capitolo introdurremo il concetto di *Energon*, una sorta di unità di misura del consumo energetico per il sistema in uso. L'Energon in qualche modo imita il ruolo del "passo algoritmico" del modello RAM, adattato qui per prendere in considerazione il profilo energetico dell'algoritmo e perché sia il più indipendente possibile dal sistema sottostante (in modo analogo al passo RAM).
- Nel quarto capitolo procederemo verso la parte teorica della nostra proposta introducendo la nozione di Ξ, la *complessità sperimentale energetica*, ispirata dalla famosa notazione Θ della complessità computazionale classica in tempo. Ξ è *composizionale* e può essere usata, così come Θ, per confrontare algoritmi dal punto di vista energetico in modo indipendente dal sistema [12]. Procederemo quindi alla validazione dell'Energon analizzando il profilo energetico di alcuni algoritmi noti (binary search ed alcuni algoritmi di sorting), mostrandone la

robustezza e la prevedibilità usando il modello RAM quando i dati siano contenuti in un singolo livello di memoria.

- Nel quinto capitolo useremo gli strumenti presentati per studiare il comportamento energetico degli algoritmi al variare delle architetture, dei pattern di accesso in memoria e dello stile di scrittura del codice, illustrando pertanto, come affermato da [9], come la relazione tra tempo di completamento e consumo energetico sia intricata. Quantificheremo alcune di queste differenze e tracceremo alcune conclusioni che speculano su possibili applicazioni della nostra teoria e metodologia per il design di algoritmi energeticamente consapevoli.

Lo stato dell'arte

I primi lavori organici che riguardano l'analisi di tecniche di scrittura software per un minor consumo energetico sono in circolazione ormai da quindici anni [13] [14] [15]. Fino ad oggi tuttavia l'attenzione è stata riservata prevalentemente ai microprocessori ed ai sistemi embedded, le misurazioni si sono sempre concentrate a livello di microistruzione per una profilazione il più possibile accurata del software.

Il primo articolo che propone un metodo sperimentale per la misurazione del consumo energetico del software è [13] in cui tuttavia, come in tutti i lavori in cui vengono presi in considerazione i costi delle singole microistruzioni, si affronta il problema cercando di analizzare il software a livello di istruzione assembler (vedi Figura 1), per poi cercare di creare tecniche di profilazione esatta del codice sorgente nel suo insieme (vedi Figura 2). La tecnica proposta da Tiwari per misurare il costo energetico delle entità di interesse (alla quale in qualche modo anche noi ci rifacciamo) è concettualmente molto semplice: realizzare un programma che consiste in una singola istruzione ripetuta un numero noto di volte in modo da rendere misurabile (una sola istruzione avrebbe una durata troppo breve per dare misurazioni attendibili) l'esperimento.

Number	Instruction	Current (mA)	Cycles
1	NOP	275.7	1
2	MOV DX,BX	302.4	1
3	MOV DX,[BX]	428.3	1
4	MOV DX,[BX][DI]	409.0	2
5	MOV [BX],DX	521.7	1
6	MOV [BX][DI],DX	451.7	2
7	ADD DX,BX	313.6	1
8	ADD DX,[BX]	400.1	2
9	ADD [BX],DX	415.7	3
10	SAL BX,1	300.8	3
11	SAL BX,CL	306.5	3
12	LEA DX,[BX]	364.4	1
13	LEA DX,[BX][DI]	345.2	2
14	JMP label	373.0	3
15	JZ label	375.7	3
16	JZ label	355.9	1
17	CMP BX,DX	298.2	1
18	CMP [BX],DX	388.0	2

Figura 1: analisi del costo delle istruzioni assembler, Tiwari, 1994

Program	Current(mA)	Cycles
; Block B1		
main:		
mov bp,sp	285.0	1
sub sp,4	309.0	1
mov dx,0	309.8	1
mov word ptr -4[bp],0	404.8	2
;Block B2		
L2:		
mov si,word ptr -4[bp]	433.4	1
add si,si	309.0	1
add si,si	309.0	1
mov bx,dx	285.0	1
mov cx,word ptr _a[si]	433.4	1
add bx,cx	309.0	1
mov si,word ptr _b[si]	433.4	1
add bx,si	309.0	1
mov dx,bx	285.0	1
mov di,word ptr -4[bp]	433.4	1
inc di, 1	297.0	1
mov word ptr -4[bp],di	560.1	1
cmp di,4	313.1	1
jl L2	405.7(356.9)	3(1)
;Block B3		
L1:		
mov word ptr _sum,dx	521.7	1
mov sp,bp	285.0	1
jmp main	403.8	3

Figura 2: Stima di un programma, Tiwari, 1994

Questo approccio è stato negli anni applicato più volte a microprocessori [16], sfruttando misurazioni con oscilloscopio o multimeter [17], oppure sfruttando le specifiche del consumo energetico a livello di microistruzione fornita dal costruttore.

Generalmente questi modelli sono alla base di sistemi predittivi del consumo energetico tramite simulazione dell'esecuzione del programma, tracciando ogni istruzione e sommando poi i consumi di ogni istruzione [18] [6] [19], simulando interi sistemi [20] [21], oppure tramite il debugging e tracing istruzione per istruzione [22].

Approcci simili sono stati usati per stimare il costo di programmi Java eseguiti in macchina virtuale, di cui tuttavia riteniamo manchi una analisi degli effetti collaterali

all'esecuzione del programma strettamente parlando, quali ad esempio il garbage-collector [23] [24].

Le tecniche sopra citate sono solitamente applicate a microprocessori, mentre l'oggetto del nostro interesse è più generale. Solo in anni più recenti sono state tentate misurazioni di sistemi workstation standard, particolarmente importante in questo contesto è il lavoro di [25]. Questo è inoltre uno dei rari casi in cui le misurazioni siano state effettuate basandosi su metodologie relativamente semplici da replicare e misurando il sistema come un'unica entità senza soffermarsi sui singoli componenti.

Lo scopo ultimo della nostra ricerca tuttavia non è solo ingegneristico (stabilire una metodologia di misurazione) ma soprattutto teorico (delineare una teoria), è in altri termini quello di analizzare in modo chiaro il rapporto esistente tra complessità algoritmica classica (tempo di completamento) e consumo energetico. In questo campo abbiamo trovato pochissimi lavori, interessante in questo senso il contributo di [26] che introduce una metrica utile per confrontare algoritmi contemporaneamente sia dal punto di vista del tempo di completamento che dell'energia assorbita.

Abbiamo trovato scarsità di letteratura riguardo il modelli di consumo energetico in ambiente multi-core. La comunità scientifica inoltre non è ancora concorde sulle interpretazioni delle misurazioni effettuate in ambiente multi-core virtualizzato mostrate ad esempio in [27], mostrate in Figura 3 e in linea con i risultati degli esperimenti da noi condotti.

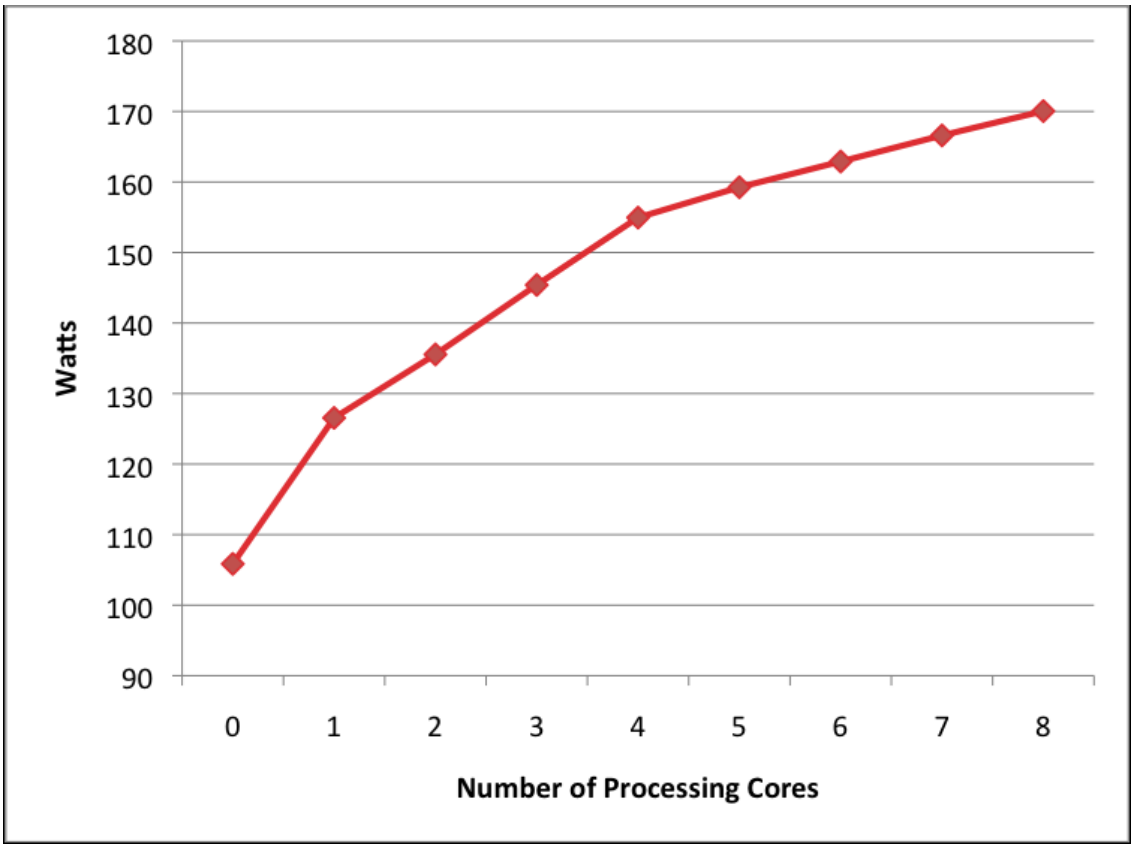

Figura 3: Potenza istantanea usata al crescere del numero di cores utilizzati, Younge, 2010

Pochi sono inoltre i casi in cui si prende in considerazione il consumo energetico del sistema operativo [28], il cui contributo nelle misurazioni dei programmi è stato a nostro avviso trascurato.

In nessuno dei lavori consultati è stata mai analizzata la relazione tra complessità computazione di un algoritmi ed il suo consumo energetico.

La Metodologia

L’ambiente di misurazione

L’ambiente di misurazione è stato ridisegnato molte volte perché sono possibili diverse soluzioni, ognuna con alcune caratteristiche positive ed altre negative.

Per favorire la ripetibilità degli esperimenti abbiamo cercato componenti hardware dotati delle seguenti caratteristiche:

- Commercialmente disponibili
- Basso costo
- Facilmente programmabili

Molte sono le possibili soluzioni tecniche, tra cui anche quella descritta in [29], la più adatta è a nostro avviso quella di sfruttare le periferiche Phidgets [30] che offrono un approccio plug-and-play all’elettronica: un microcontrollore dotato di interfaccia USB può essere collegato con molteplici sensori dotati di interfaccia standard. È stato usato l’amperometro Phidgets 1122 che ha un range di 30A e una risoluzione di 0,04A in Corrente Alternata (Figura 4). L’errore tipico oscilla tra 1% e 2% con un massimo di 5%. Offre inoltre numerose librerie di programmazione che coprono praticamente tutti i sistemi operativi e ambienti di programmazione. È stata usata l’interfaccia .NET e abbiamo sviluppato il framework di misurazione usando C#. Altri amperometri possono essere usati al posto di quello usato purché offrano caratteristiche analoghe.

Figura 4: sistema Phidgets, controllore e amperometro

Dato questo hardware abbiamo proceduto a misurare la corrente applicando amperometri alle linee di alimentazione in corrente continua della scheda madre intercettando i cavi dall’alimentatore. Questo approccio è stato seguito credendo che la circuiteria dell’alimentatore, che trasforma la corrente alternata in continua, avrebbe ridotto la capacità di lettura dell’amperometro. Così facendo pensavamo inoltre, seguendo [4], di essere in grado di sperimentare i contributi delle diverse componenti del sistema, quali CPU, memoria, disco fisso.

Sebbene questo approccio abbia dato letture sofisticate e precise abbiamo deciso di abbandonarlo, per misurare i consumi in corrente continua infatti è necessario intervenire all’interno del calcolatore ed inserire numerosi amperometri, rendendo costoso e difficile replicare l’ambiente di misurazione. Inoltre l’eccessivo dettaglio (all’interno del calcolatore sono presenti decine di linee di alimentazione) non aggiungeva informazioni utili al nostro scopo addirittura nascondendo l’energia dissipata dall’alimentatore stesso. Abbiamo quindi deciso di tentare un approccio molto più semplice (ma assente in letteratura): misurare il consumo energetico a monte dell’alimentatore, in corrente alternata. I molti esperimenti seguiti hanno mostrato che questo approccio fornisce misurazioni affidabili pur essendo estremamente semplice da realizzare, avendo un costo minimo e non richiedendo la modifica di alcun elemento del sistema misurato.

Abbiamo definito una metodologia di misurazione composta da due sistemi sui quali vengono eseguiti diversi software (Figura 5): un *sistema di misurazione* che esegue un software di misurazione, responsabile dell’acquisizione dei dati dall’amperometro; un *sistema di controllo* che tramite un *software di controllo* avvia l’algoritmo oggetto di

misurazione (*programma misurato*) ed invia messaggi TCP al *sistema di misurazione* per segnalare l'inizio e la terminazione del programma misurato.

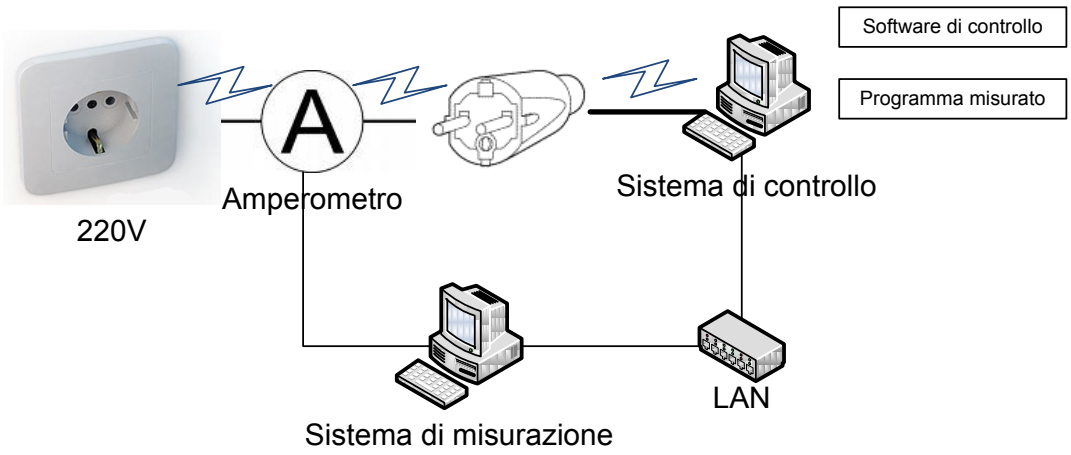

Figura 5: L'ambiente di misurazione

È stato verificato che l'uso di connessioni TCP e le operazioni necessarie alla gestione del *programma misurato* non influiscono significativamente sulle misurazioni ottenute (per dettagli si veda l'Appendice "Misurare il costo del sistema di misurazione"). Questo approccio è risultato essere sufficientemente solido da permetterci di ignorare le micro variazioni dovute all'*ecosistema*, purché gli esperimenti non vengano eseguiti in parallelo ad altri programmi *significativi* sul *sistema di controllo* (CPU sostanzialmente idle). La prossima sezione approfondirà il concetto di *ecosistema*.

In accordo al metodo scientifico, per favorire la ripetibilità degli esperimenti effettuati, i due software usati sul *sistema di misurazione* e sul *sistema di controllo* sono rilasciati con licenza Open Source e sono disponibili su [31].

Il programma in esecuzione sul *sistema di misurazione* esegue un subsampling dei dati ricevuti dall'amperometro stabilizzando il sample rate a 10Hz e riducendo così i possibili errori sulla misurazione del tempo di completamento del *programma misurato*. I programmi misurati devono quindi avere una durata di almeno 100msec, questo requisito ci ha portato a eseguire un numero di volte conosciuto gli algoritmi troppo veloci da misurare, ispirandoci a [13]. L'amperometro invia letture degli ampere usati dal *sistema di controllo* permettendoci quindi di ricavare la Potenza istantanea usata.

$$P = V * I$$

Conoscendo infatti la corrente (I) ed il voltaggio (in Italia 220V) si ricava la potenza (in Watt). Abbiamo tralasciato il fattore di potenza $\cos(\Phi)$ nella conversione, con una conseguente sovrastima della potenza di un fattore costante.

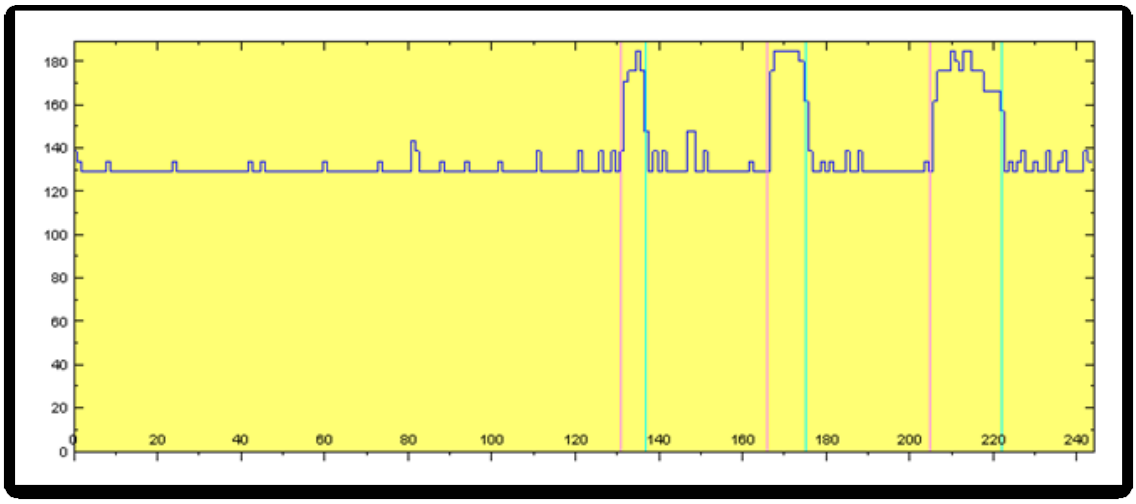

Figura 6: misurazione del sistema operativo idle e con algoritmo in esecuzione

Figura 6 mostra la potenza istantanea consumata (in Watt) su un sistema dotato di 8 core, prima in stato idle e poi durante l'esecuzione di un programma che effettua una scansione lineare di un array di grandi dimensioni (marcato da linee rosse e verdi). Il sistema operativo, come mostrato nel dettaglio da [32], ha un assorbimento energetico stabile nel tempo. È inoltre interessante notare che un uso a pieno carico degli otto core comporta un incremento della potenza istantanea di circa un 40%, suggerendo il fatto che i sistemi in stato di idle consumino molta energia, come già affermato da [33].

Moltiplicando la Potenza Istantanea media (in Watt) usata durante l'esecuzione dell'algoritmo per il tempo di completamento (in secondi) dell'algoritmo otteniamo l'Energia (in Joule) usata dal sistema di elaborazione per completare l'esecuzione dell'algoritmo.

$$E = P * T$$

È stato esaminato il consumo energetico di un algoritmo più interessante: binary search. Figura 7 mostra il consumo energetico espresso in Joule ottenuti iterando ogni esecuzione del binary search 2^{20} volte (per portare il completamento dell'algoritmo ad

una durata misurabile). È da notare come la nostra semplice metodologia e gli apparati hardware scelti riescano a mostrare una relazione evidente tra il comportamento energetico di questo algoritmo e la sua complessità computazionale, che essendo $\Theta(\log n)$ ci aspettavamo difficile da apprezzare. La funzione logaritmica si adatta al grafico del consumo energetico.

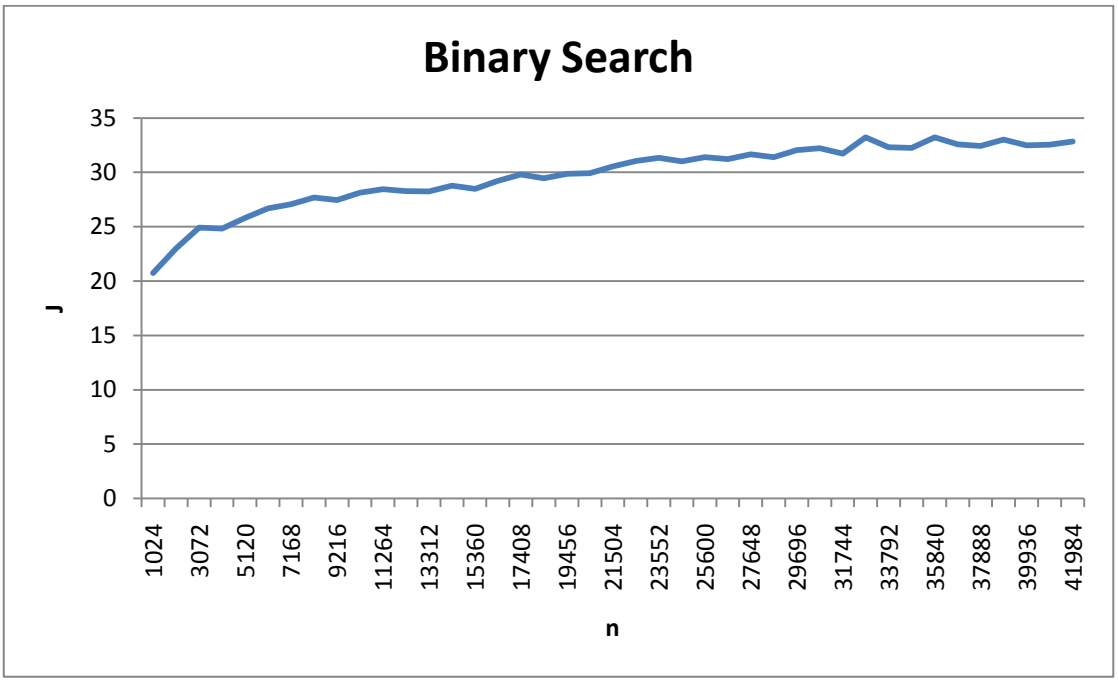

Figura 7: Consumo energetico del Binary Search in J

Il software eseguito sul *sistema di controllo* si occupa di avviare il programma oggetto di misurazione e di informare il *sistema di misurazione* tramite messaggi TCP. Il programma oggetto di misurazione deve avere una struttura simile a quella mostrata in Figura 8.

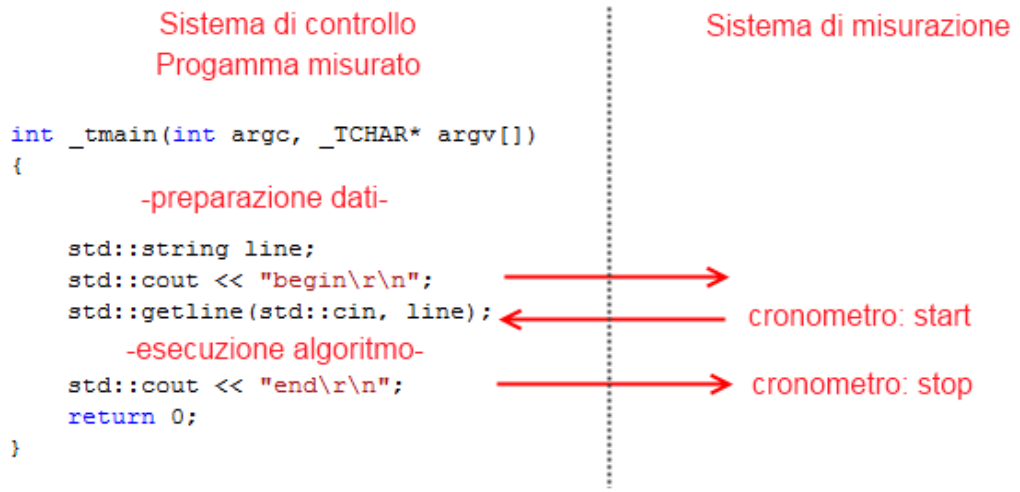

Figura 8: Struttura del programma misurato

Si deve comunque osservare come esprimendo l’energia in J sia possibile speculare sul consumo energetico di un particolare sistema ma non si possano confrontare misurazioni effettuate su sistemi diversi. Figura 9 mostra la differenza delle misurazioni effettuate su due sistemi diversi eseguendo lo stesso algoritmo con le stesse dimensioni di input. Per la formulazione di un modello convincente riteniamo sia di fondamentale importanza il poter confrontare misurazioni di uno stesso algoritmo prese su sistemi diversi, è quindi necessaria una unità di misura che renda i risultati invarianti rispetto al sistema sottostante l’esperimento. Questa unità di misura verrà introdotta nella prossima sezione.

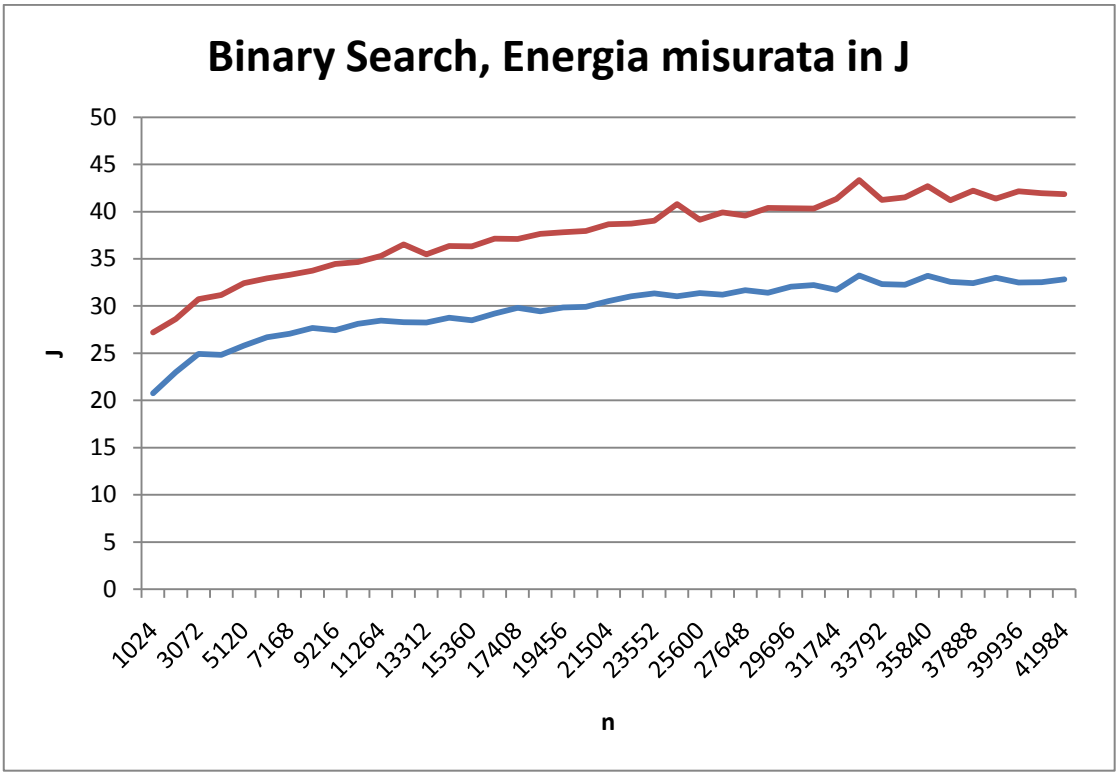

Figura 9: differenza delle misurazioni dello stesso algoritmo su due sistemi simili

Energon

È noto come le misurazioni prese in un ambiente controllato differiscano rispetto a quelle prese nel mondo reale. Nel nostro caso il sistema osservato è una sorta di *ecosistema*, che include processi del sistema operativo e servizi attivati da eventi esterni e che interagiscono con altre applicazioni. Vogliamo misurare i cambiamenti che avvengono durante l'esecuzione del nostro programma, distinguendoli dal rumore di fondo causato dall'ecosistema in sé.

Il primo tentativo è stato quello di misurare la potenza istantanea media del sistema senza il programma in esecuzione e sottrarre questo valore dalla potenza istantanea media del sistema con il programma in esecuzione, moltiplicando poi il valore ottenuto per il tempo di completamento per ottenere l'energia usata dal programma, questo procedimento è equivalente a calcolare l'area delle regioni del grafico in rosso in Figura 10. Riteniamo questo approccio tuttavia errato perché non abbiamo controllo né conoscenza dei processi del sistema operativo attivi prima, durante o dopo l'esecuzione del programma, e non riteniamo sia utile, come fatto in precedenza da altri, cercare un livello di dettaglio tale da scomporre i contributi energetici dei vari processi del sistema operativo. Il sistema sul quale viene eseguito il programma è un ecosistema complesso.

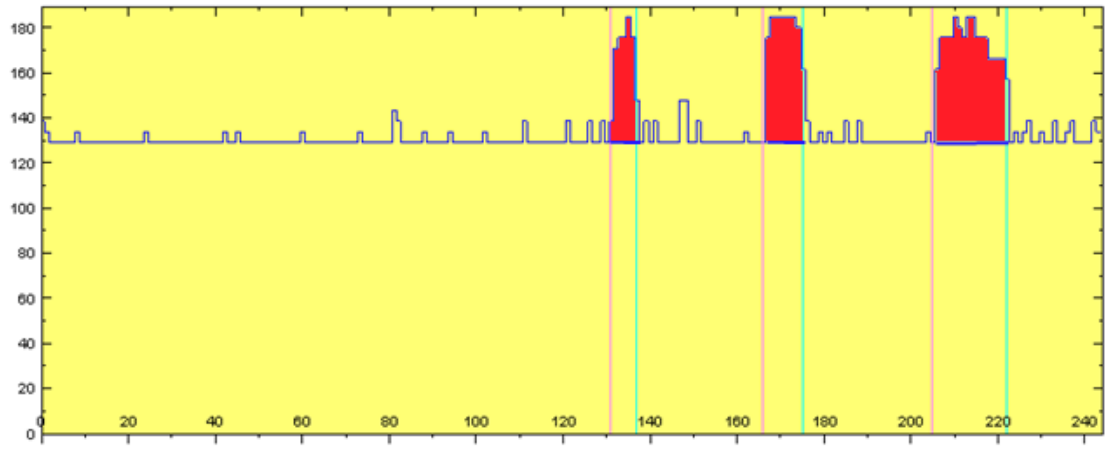

Figura 10: Ipotesi di energia usata dal programma

Per aggirare questo problema abbiamo tratto ispirazione dall'approccio usato dagli ingegneri biomedici per la misurazione delle colture cellulari [34]. Quando i biologi misurano l'attività delle cellule su un terreno di coltura, ad esempio durante il testing di

medicinali, effettuano prima una misurazione su un sistema di riferimento e dopo aver effettuato il trattamento ripetono la misurazione. I risultati sono quindi normalizzati rispetto alla prima misurazione che assume quindi il ruolo di metro. Due sono gli aspetti per noi interessanti in questo approccio: le misurazioni di riferimento sono effettuate perché è virtualmente impossibile riprodurre i parametri ambientali esatti (quali ad esempio temperatura, umidità, etc.); ed il riferimento include le celle, non solo il terreno, perché le celle che interagiscono con l'ecosistema potrebbero alterare i parametri del terreno indipendentemente dal trattamento oggetto di misurazione.

Abbiamo trovato forti analogie tra il nostro framework e l'ecosistema cellulare. In particolare il fatto che un sistema operativo potrebbe modificare l'ambiente di esecuzione del programma quando questo verrà avviato (ecosistema). Abbiamo quindi introdotto un *programma di riferimento* nella nostra metodologia sperimentale che viene eseguito nello stesso ecosistema in cui viene eseguito il programma oggetto di misurazione. Il costo energetico di questo programma di riferimento viene quindi usato come metro per normalizzare i risultati degli esperimenti. Sono possibili scelte diverse per questo metro, che prendano in considerazione le diverse caratteristiche architetturali e dei servizi del sistema, e tali metri alternativi potrebbero portare a risultati diversi, oggetti di ulteriore studio. In questo lavoro proponiamo il programma *Energon*, che è concepito per essere il più semplice programma possibile, non influenzabile dal compilatore, che usi la CPU (o un core della CPU) al 100% per un certo numero (1G) di istruzioni, subito prima che l'esperimento sia avviato sul programma misurato.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int a=0;
    int i = 0;
    std::string line;
    std::cout << "begin\r\n";
    std::getline(std::cin, line);

     __asm
     {
      mov ecx, 40000000h
      mov eax, 0
loop0:
      inc eax
      loop loop0
      mov a, ebx
     }

    std::cout << "end\r\n";
    return 0;
}
```

programma misurato

1G cicli

Figura 11: il codice sorgente di Energon

Figura 11 mostra il semplice codice sorgente C++ del programma Energon. Consiste di una istruzione assembler inserita in un loop. Il restante codice C++ è usato per comunicare col software di controllo che a sua volta comunicherà con il sistema di misurazione per avviare il timer. In questo modo programmi complessi possono escludere la fase di setup dei dati (quali ad esempio l'allocazione delle risorse) dalla misurazione.

Il codice può essere facilmente adattato a differenti architetture. In questa fase abbiamo deliberatamente evitato di usare memoria poiché la comunicazione tra memoria e CPU avviene in modo asincrono e potrebbe introdurre cicli idle non voluti nella CPU, in modo dipendente dalle gerarchie di memoria. Discuteremo questo aspetto più approfonditamente nel capitolo "Considerazioni" sezione "Pattern di accesso in memoria".

Quando misuriamo un algoritmo la sua misura in Energon (usiamo la lettera greca ε per denotare 1 Energon) è calcolata come il rapporto tra l'energia assorbita dall'algoritmo e quella assorbita dal programma Energon sullo stesso sistema. Questo rende la misurazione più stabile e robusta perché include il consumo dell'ecosistema durante l'esecuzione del programma misurato. Come già accennato, le ogni test viene eseguito

più volte (tutti i test in questo lavoro sono stati eseguiti almeno 30 volte) e viene poi calcolato il consumo medio del programma Energon e del programma oggetto di misurazione. Abbiamo ottenuto una deviazione standard del 2,8% sull'energia richiesta dall'Energon. Questo dato si riferisce a misurazioni effettuate su un AMD Athlon 64 X2 Dual Core 4200++, 2.20 GHz 2 GB RAM, sistema operativo Windows Vista Home Edition; sono stati misurati 177,25J medi con una deviazione standard di 3,46J.

Abbiamo effettuato misurazioni sull'algoritmo Binary Search su tale sistema. Abbiamo poi ripetuto la misurazione dell'Energon su un altro sistema, sempre famiglia di processore AMD ma con diverse componenti e prestazioni, abbiamo quindi misurato l'algoritmo Binary Search su questo secondo sistema. Figura 9 mostra i risultati espressi in J. Si noti come le curve si assomiglino senza però sovrapporsi. Portando però le misurazioni in Energon otteniamo un'ottima sovrapposizione dei risultati come mostrato in Figura 12.

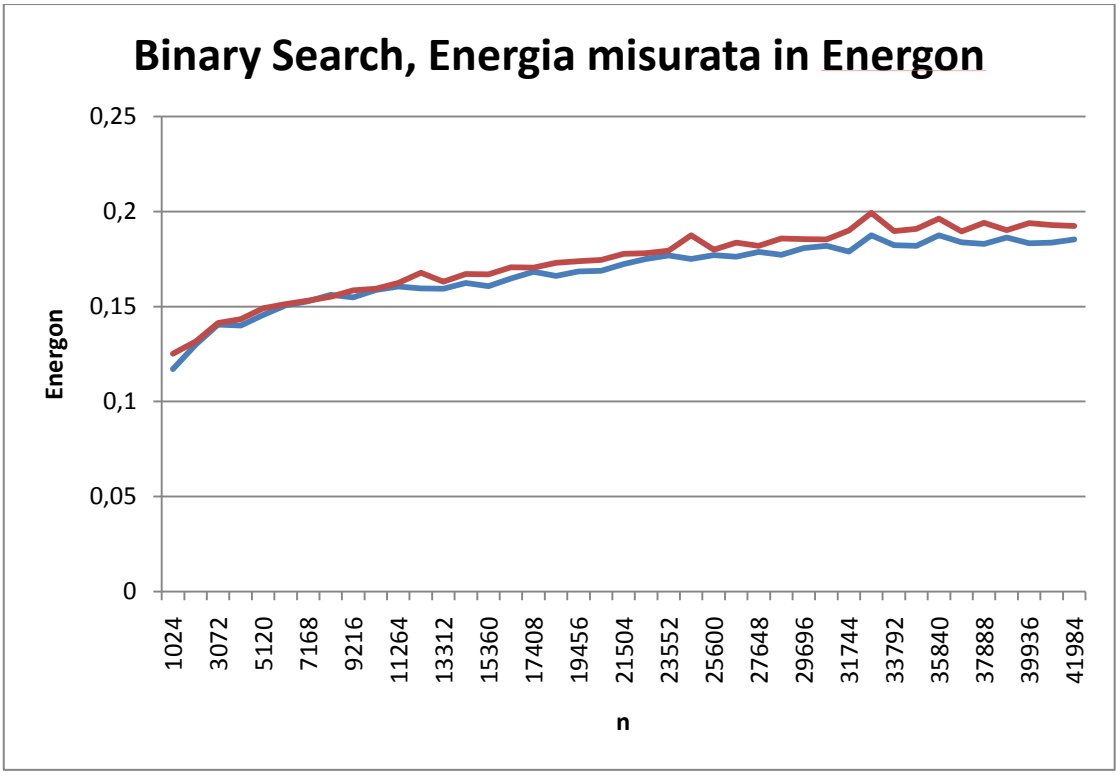

Figura 12: Energia del Binary Search su due sistemi simili misurata in Energon.

L'uso di Energon come unità di misura, o comunque di un programma usato come riferimento, è quindi uno strumento che permette di astrarre i risultati rispetto al sistema usato, almeno su architetture simili.

Complessità computazionale sperimentale

In questo capitolo indagheremo sulla relazione tra tempo di completamento e consumo energetico. Ci aspettiamo di trovare un rapporto molto stretto, essendo $E = P * T$. Nelle sezioni "Composizionalità" e "Algoritmi di ordinamento" mostreremo risultati sperimentali che confermano questa ipotesi. Nel capitolo "Considerazioni" vedremo invece casi in cui l'analisi della complessità temporale non è sufficiente a descrivere il comportamento energetico degli algoritmi.

Iniziamo definendo una notazione utile a caratterizzare il comportamento energetico degli algoritmi al variare della dimensione di input. Abbiamo chiamato questa notazione Ξ ed abbiamo scelto di seguire un approccio il più vicino possibile alla teoria della complessità classica per poter mettere in evidenza similitudini e differenze rispetto a Θ. Abbiamo comunque voluto colmare la distanza che separa questo un approccio da quello che avrebbe seguito un fisico definendo anche ξ, basata sul fitting dei dati.

Ξ

Introduciamo una notazione per la nostra teoria della complessità sperimentale ispirata alla tradizionale notazione Θ. Nel nostro caso però non possiamo sfruttare il comportamento asintotico poiché per definizione le misurazioni possono essere eseguite solo su insiemi di dati finiti.

Definizione Ξ

Dato un insieme di misurazioni A espresse in Energon che rappresentano la complessità sperimentale di un algoritmo alg rispetto ad un insieme di dati di input nell'intervallo $[a, b]$, diciamo che l'insieme di dati appartiene a $\Xi(f(n))$ sull'intervallo $[a, b]$ se esistono due costanti k_1 e k_2 tali che:

- $\forall x \in [a, b]. data(x) \in A \Longrightarrow k_1 f(x) \leq data(x) \leq k_2 f(x)$
- $\exists m_1, m_2 \in A, x_1, x_2 \in [a, b] \;\; t.c. \;\; k_1 f(x_1) = m_1 \wedge \;\; k_2 f(x_2) = m_2$

e scriveremo che $A \in \Xi_a^b(f)$ rispetto a ε, oppure più semplicemente che A è $\Xi(f)$.

Questa definizione cerca di catturare l'idea che le misurazioni possono essere racchiuse da una funzione pesata con due costanti, con il requisito aggiuntivo che ogni funzione pesata deve passare per l'insieme dei dati in modo da scegliere i valori delle costanti che minimizzano la distanza.

Come esempio Figura 13 mostra come Binary Search appartenga a $\Xi(n \log(n))$.

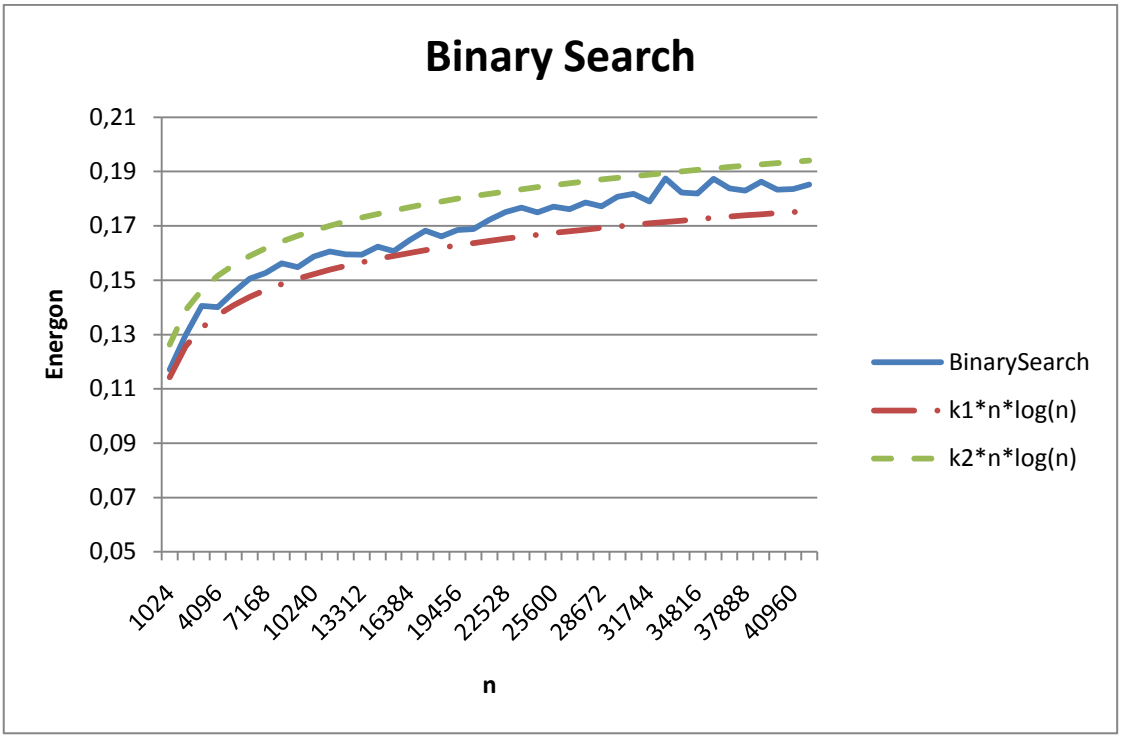

Figura 13: Binary search è Ξ(log n)

Questa definizione permette di scegliere i migliori valori di k_1 e k_2 per una data funzione ma essendo l'intervallo finito è possibile trovare valori delle costanti tali da far appartenere i dati a Ξ di qualunque funzione.

Introduciamo quindi alcune notazioni per caratterizzare aspetti rilevanti di Ξ.

Definizione Ξ preferita

Dato un insieme di dati A su un intervallo $[a, b]$ e due funzioni f e g diciamo che f è Ξ preferita rispetto a g se $A \in \Xi(f)$ e $A \in \Xi(g)$ ma f ha uno scarto quadratico medio rispetto ai dati di A minore rispetto a g.

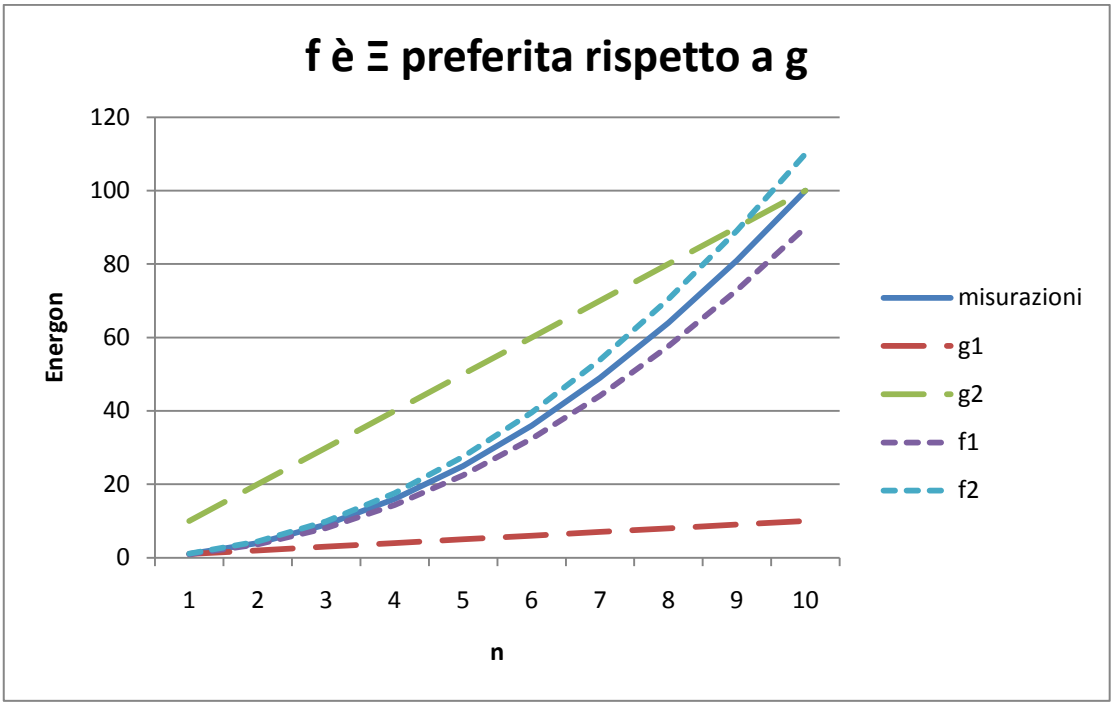

Figura 14: f è Ξ preferita rispetto a g

Definizione Ξ ideale

Una funzione f è detta Ξ ideale rispetto ad un insieme di dati A, un intervallo $[a, b]$, un insieme di funzioni F se $A \in \Xi(f)$, $f \in F$ e se esiste $g \in F$ tale che g è Ξ preferita rispetto a f allora $f = g$.

In altre parole f è la funzione che tra tutte quelle presenti in un insieme di funzioni F minimizza lo scarto quadratico medio con i dati.

Definizione ξ

Una volta individuata la funzione Ξ ideale rispetto ad un insieme di dati A, chiamiamo $\bar{k}$ il valore di k che minimizza lo scarto quadratico medio di f rispetto ai dati.

Chiamiamo inoltre $\xi(x)$ la funzione $\bar{k}f(x)$.

La funzione $\xi(x)$ può essere quindi usata come approssimazione e previsione del costo computazione dell'algoritmo le cui misurazioni sono espresse dall'insieme A.

Relazione tra Ξ e Θ

Queste definizioni cercano di catturare il comportamento dei dati misurati descrivendoli in termini di funzioni. Ad esempio abbiamo verificato che merge-sort è $\Xi(n\log(n))ideale$ tra le funzioni generalmente prese come modello per la complessità computazionale ($\log n$, n, $n\log n$, n^2, etc.).

Tra Ξ e Θ ci sono ovvie similitudini, molto spesso infatti risulterà che se la complessità computazione in tempo di un algoritmo è $\Theta(f)$ allora i dati delle sue misurazioni saranno $\Xi(f)$.

Esistono tuttavia alcune differenze fondamentali tra Ξ e Θ:

- Θ cattura il comportamento asintotico di un algoritmo mentre Ξ individua una funzione che ne sintetizza il consumo energetico in un intervallo finito.
- Non è possibile semplificare la funzione Ξ (essendo applicata ad un insieme finito di valori), un esempio è $\Xi_0^1(x^2+x) \neq \Xi_0^1 x^2$, infatti nell'intervallo [0,1] x prevale su x^2.

Composizionalità

Prendiamo l'algoritmo f, disegnato per essere di semplice comprensione e la cui complessità algoritmica sia semplice da analizzare. Il codice sorgente di f è il seguente:

```
// f has complexity = log n
void f(int* tab, int n)
{
      int i=n-1;
      // an O(log(n)) loop
      while(i>1)
      {
            int a;
            // an O(1) job
       __asm
       {
        mov ecx, 1000000h
        mov eax, 0
loop0:
        inc eax
        loop loop0
        mov a, ebx
       }
            tab[i]=a;
```

```
            i /= 2;
        }
}
```

Si verifica che $f \in \Xi\lceil \log_2 n \rceil$ e che $\lceil \log_2 n \rceil$ è Ξ^{id}. Figura 15 mostra come f segua un comportamento energetico atteso di $\lceil \log_2 n \rceil$, simile alla complessità algoritmica dedotta analizzando il codice sorgente di f.

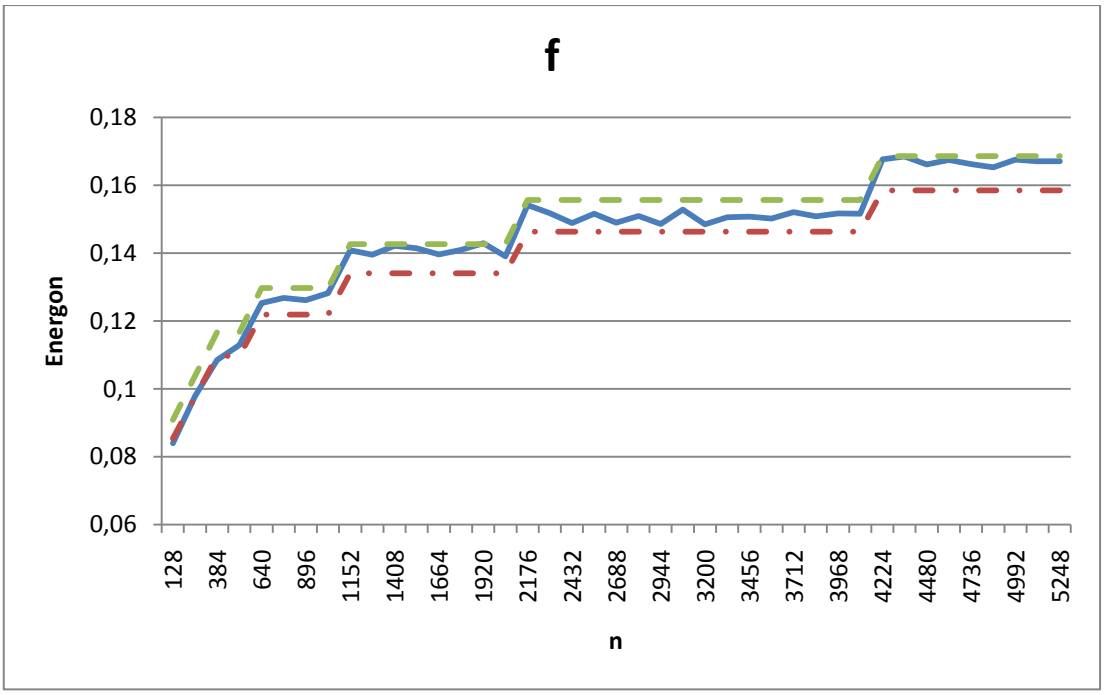

Figura 15: f e la funzione Ξ che la descrive

Figura 16 mostra invece la funzione ξ di f ricavata numericamente:

$$\xi = \overline{k_f}\lceil \log_2 n \rceil, \qquad \overline{k_f} = 0{,}012578125$$

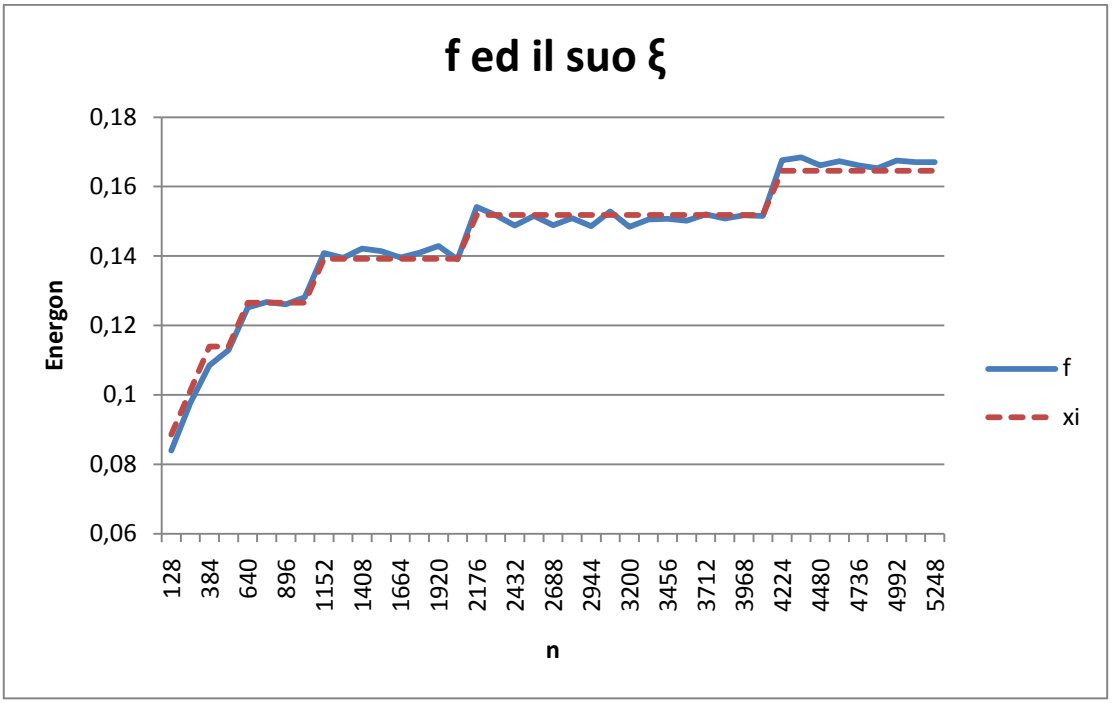

Figura 16: f ed il suo ξ

Prendiamo ora l'algoritmo g il cui codice sorgente è:

```
// g has complexity = n
void g(int* tab, int n)
{
	// an O(n) loop
	for (int i=0; i<n/2; i++)
	{
		int a;
		// an O(1) job
	  __asm
	  {
	   mov ecx, 100000h
	   mov eax, 0
loop0:
	   inc eax
	   loop loop0
	   mov a, ebx
	  }
		tab[i]=a;
	}
}
```

Si verifica che $g \in \Xi n$ e che n è $\Xi - ideale$, Figura 17 e Figura 18 mostrano rispettivamente i grafici di Ξ e ξ:

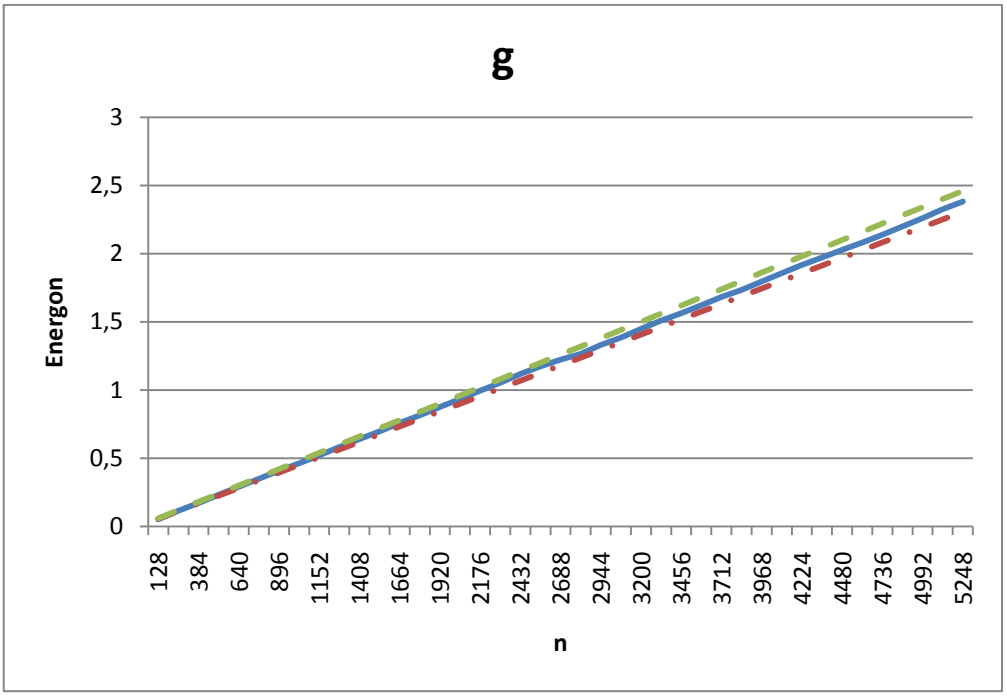

Figura 17: g appartiene a Ξ di n

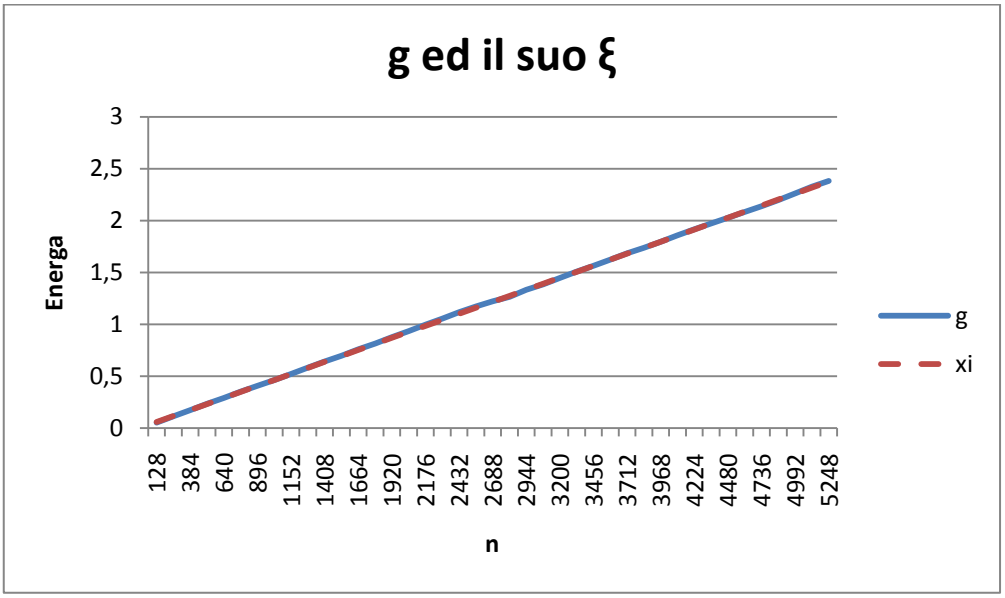

Figura 18: g ed il suo ξ

$$\xi = \overline{k_g} n, \qquad \overline{k_g} = 0{,}00045263671875$$

$\xi_{f+g} = \xi_f + \xi_g$

Prendiamo ora l'algoritmo $f + g$ il cui codice sorgente consiste nell'invocare prima f poi g sullo stesso input.

Risulta che

$$f + g \in \Xi^{id}(n + \lceil \log_2 n \rceil)$$

Mostriamo ora come sia possibile usare $\overline{k_f}$ e $\overline{k_g}$ per ottenere uno ξ_{f+g} estremamente rappresentativo di $f + g$, definiamo:

$$\xi_{f+g} = \overline{k_f}\lceil \log_2 n \rceil + \overline{k_g} n$$

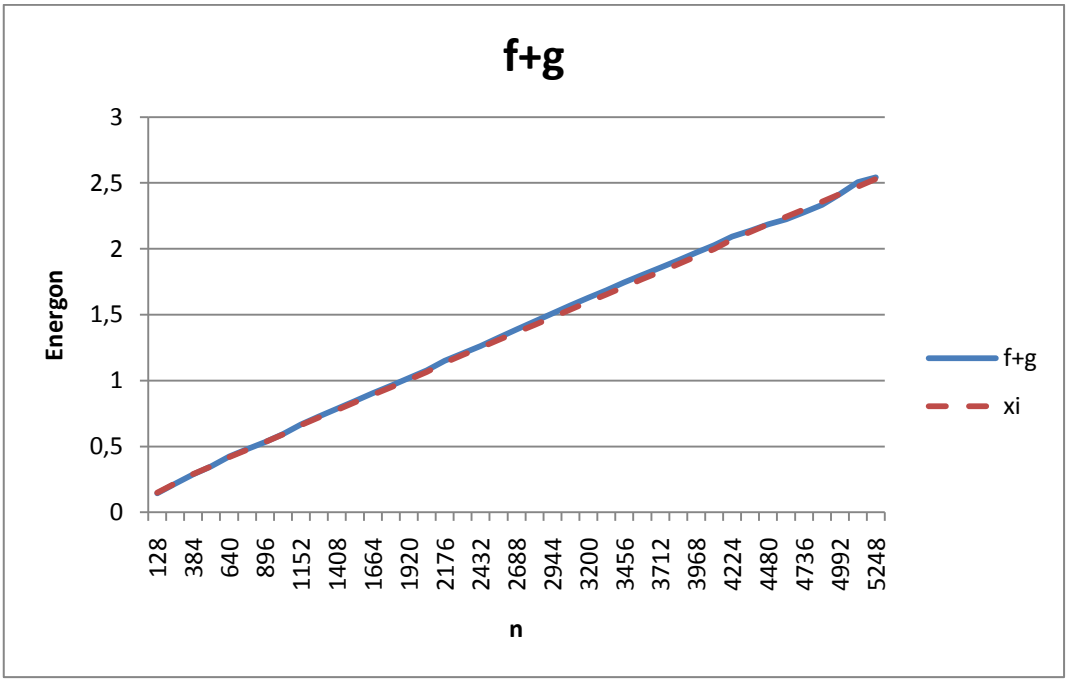

Figura 19: ξ di f e g eseguite sequenzialmente è facilmente calcolabile

L'accuratezza di ξ_{f+g} così costruito è molto alta.

$$\xi_{f+g} = \xi_f + \xi_g$$

Nota: da questa proprietà si ricava che

$$\xi_{af} = a\xi_f$$

Cioè ξ di una funzione ripetuta a volte vale $a\xi_f$, Infatti:

$$\xi_{af} = \xi_{f_1+f_2+\cdots+f_a} = \xi_f + \xi_f + \cdots \xi_f = a\xi_f$$

Questa proprietà è stata usata durante gli esperimenti per portare programmi a durate misurabili.

$\xi_{f\circ g} = \xi_f + h(n) * \xi_g$

Prendiamo ora l'algoritmo $f \circ g$ il cui codice sorgente è:

```
// f_dot_g has complexity = n log n
void f_dot_g(int* tab, int n)
{
	int i=n-1;
	while(i>1)
	{
		int a;
		// an O(1) job
	 __asm
	 {
	  mov ecx, 1000000h
	  mov eax, 0
loop0:
	  inc eax
	  loop loop0
	  mov a, ebx
	 }
		tab[i]=a;
		i /= 2;
		g(tab, n);
	}
}
```

È possibile individuare $\xi_{f\circ g}$ con ottima approssimazione partendo da ξ_f, ξ_g e ricavando $h(n)$.

Analizzando il codice della funzione di esempio `void f_dot_g(int* tab, int n)` possiamo suddividere il ciclo while (eseguito $\log_2 n$ volte) in due parti:

1) Il codice di f
2) La chiamata a g

La prima parte ha costo costante e viene eseguita $\log_2 n$ volte, il costo complessivo è quindi lo stesso di f, cioè ξ_f.

La seconda parte consiste nella sola invocazione della funzione g (che sappiamo avere costo ξ_g), in questo caso viene eseguita $\log_2 n$ volte, più generalmente viene eseguita $h(n)$ volte dove $h(n)$ è una funzione in qualche modo legata a Ξ_f. Il costo complessivo della seconda parte è quindi $h(n) * \xi_g$.

Se ne deduce quindi che:

$$\boldsymbol{\xi_{f\circ g} = \xi_f + h(n) * \xi_g}$$

In cui $h(n)$ è da ricavare analizzando il codice sorgente oppure numericamente sulle misurazioni con tecniche simili a quelle usate per ricavare ξ_f e ξ_g.

Nel nostro esempio:

- $\xi_f = 0{,}013 * \lceil \log_2 n \rceil$: dato noto
- $\xi_g = 0{,}00045 * n$: dato noto
- $h(n) = 0{,}94 * \lceil \log_2 n \rceil$: funzione da ricavare, in questo caso $\lceil \log_2 n \rceil$ è ricavato analizzando il codice sorgente e 0,94 è ricavato numericamente cercando il valore che minimizza lo scarto quadratico medio.

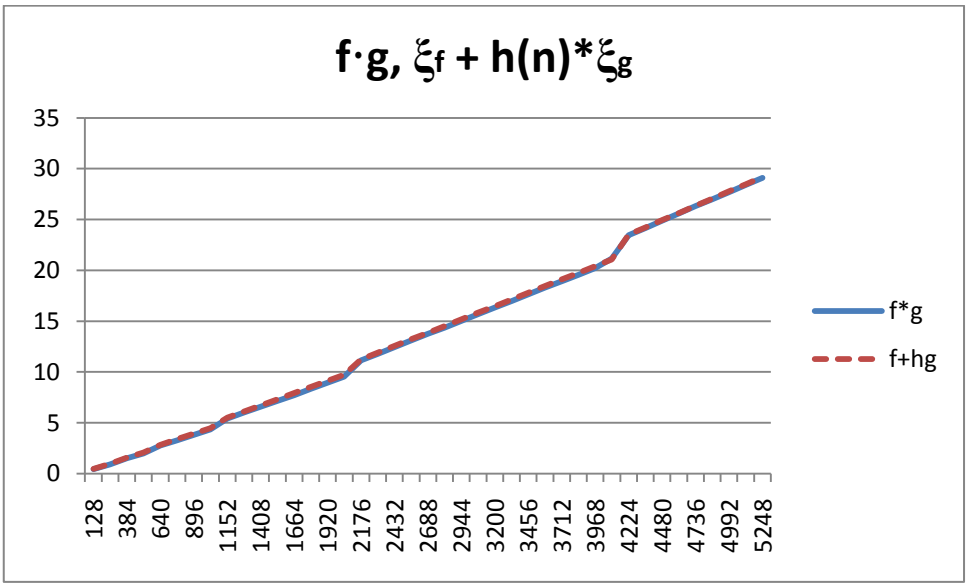

Figura 20: ξ della composizione di f e g richiede l'analisi del sorgente

Considerazioni sulla composizionalità

Le funzioni f e g sono solo un esempio e sono infatti disegnate in modo da essere le più semplici possibili, le cui complessità computazionali fossero immediate da calcolare. Per validare questi risultati preliminari sarà necessario ripetere i test su un vasto insieme di funzioni, dotate di diverse complessità computazionali. Il nostro scopo era tuttavia soltanto quello di mostrare con sufficiente chiarezza la solidità delle notazioni introdotte e di alcune loro proprietà. Queste notazioni saranno utili nei prossimi capitoli per descrivere gli algoritmi misurati ed alcune proprietà dimostrate ($\xi_{af} = a\xi_f$) sono alla base del sistema di misurazione usato (gli algoritmi troppo veloci sono ripetuti un numero noto di volte per essere portati ad un tempo misurabile).

Algoritmi di ordinamento

In questa sezione valideremo la robustezza del metodo di misurazione proposto investigando il comportamento del consumo energetico di alcuni algoritmi di ordinamento al crescere della dimensione di input.

Abbiamo studiato tre algoritmi di ordinamento noti: merge-sort, heap-sort e quicksort[11]. Questi algoritmi hanno diverse caratteristiche per quanto riguarda il caso pessimo temporalmente, o la quantità di spazio richiesto. In tutti i casi è stata misurata solo la fase di ordinamento, ignorando la creazione dell'array e la popolazione con dati casuali.

Merge-sort è un algoritmo di ordinamento ottimale basato sul confronto che richiede tempo $\Theta(n \log n)$ sia nel caso medio che in quello pessimo. Come mostrato in Figura 21 l'energia consumata dal merge-sort come funzione della dimensione di input si adatta perfettamente alla complessità temporale asintotica.

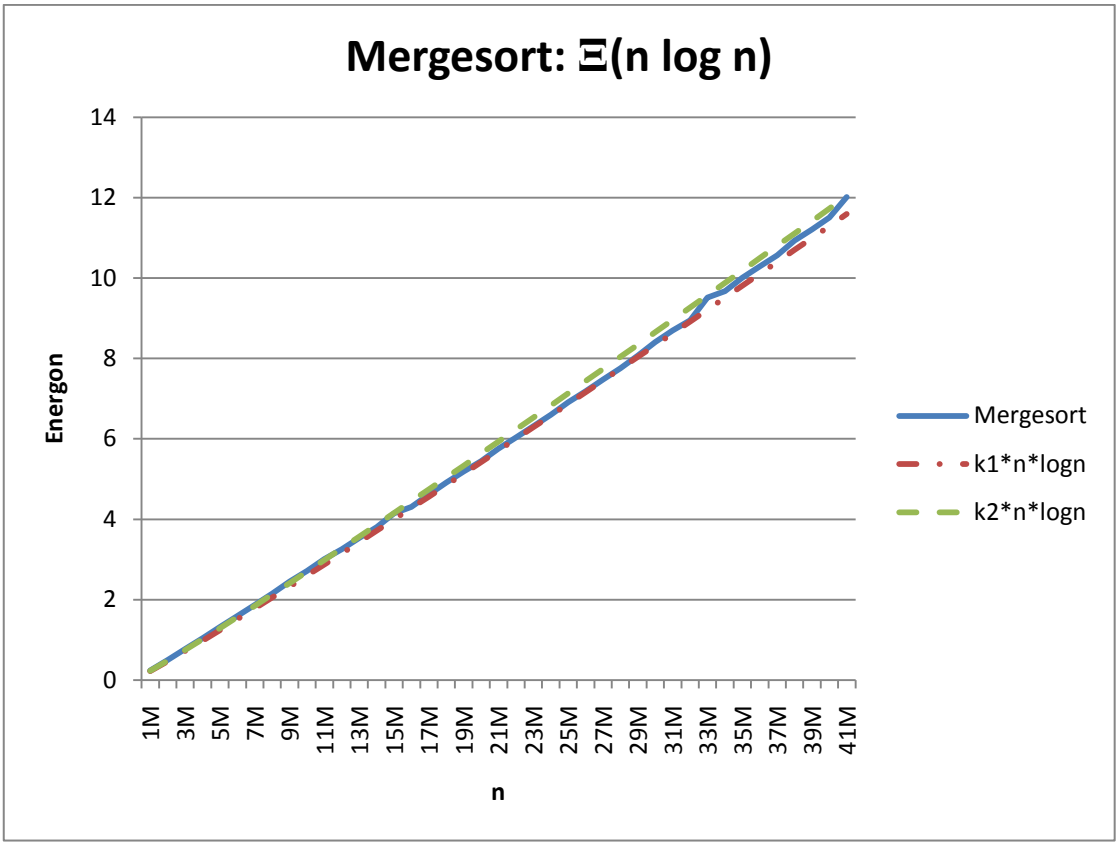

Figura 21: Merge-sort

Misurando lo heap-sort (Figura 22) abbiamo notato che richiede meno della metà dell'energia del merge-sort. Riteniamo che questo comportamento sia dovuto al pattern casuale di accesso in memoria, che implica un maggior tempo di attesa da parte della CPU per i cache-miss. Dati questi risultati, la sezione "Pattern di accesso in memoria" analizzerà nel dettaglio l'impatto non trascurabile del pattern di accesso alla memoria per la profilazione energetica degli algoritmi. Questo è tuttavia un caso interessante nel quale l'analisi dei dati sperimentali ha mostrato un dettaglio interessante non direttamente ricavabile dall'analisi della complessità algoritmica.

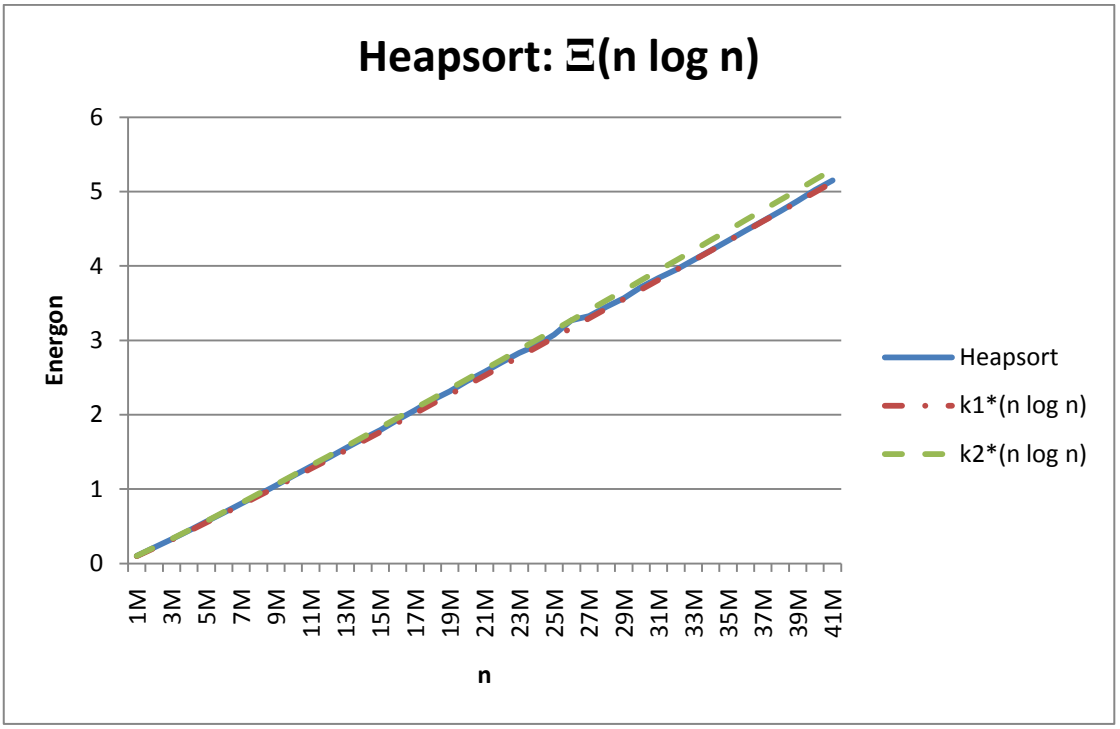

Figura 22: Heapsort

È noto come la complessità temporale del quicksort sia $\Theta(n \log n)$ nel caso medio e $\Theta(n^2)$ nel caso pessimo che si verifica per particolari configurazioni dell'array di input che provocano invocazioni ricorsive effettuate su dimensioni di input non bilanciate. Per costruire l'input in modo da cercare di ricadere sempre nel caso medio abbiamo (in linea con gli esperimenti sugli altri algoritmi) randomizzato l'array di input e ripetuto gli esperimenti per almeno 30 iterazioni. Abbiamo usato il generatore di numeri pseudo-casuali del C, che è risaputo essere lontano dall'ottimale, inizializzandolo con l'istruzione "`srand((unsigned)time(0));`" prima dell'inizio della randomizzazione dell'array. È quindi interessante notare in Figura 23 e Figura 24 come Ξ ideale non sia n^2 né $n \log n$, Figura 25 mostra invece come $f(n) = n \log n + \frac{n^2}{256000}$ riesca a descrivere molto fedelmente l'energia consumata. Non siamo stati in grado di riprodurre una funzione sufficientemente precisa usando la sola $n \log_x n$, neanche variando la base x (come suggerito dall'analisi del caso medio in [11]). Il caso pessimo deve quindi essere incluso, seppur pesato con una costante moltiplicativa che ne riduce fortemente l'impatto, nella funzione usata per esprimere il consumo energetico del quicksort.

Questo risultato è importante perché dalla teoria computazionale ci aspettavamo un consumo energetico esprimibile con la sola $n \log n$. Questo dimostra quindi come il

rapporto tra tempo ed energia sia molto più intricato di quanto ci si aspetti. Sarebbe comunque utile verificare questo risultato usando input ottimale, usando input pessimo ed usando generatori di numeri casuali più affidabili.

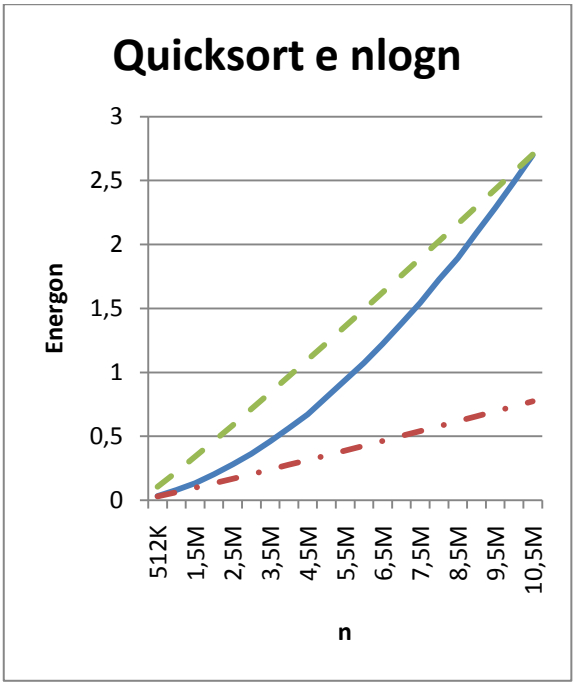

Figura 23: Quicksort e nlogn

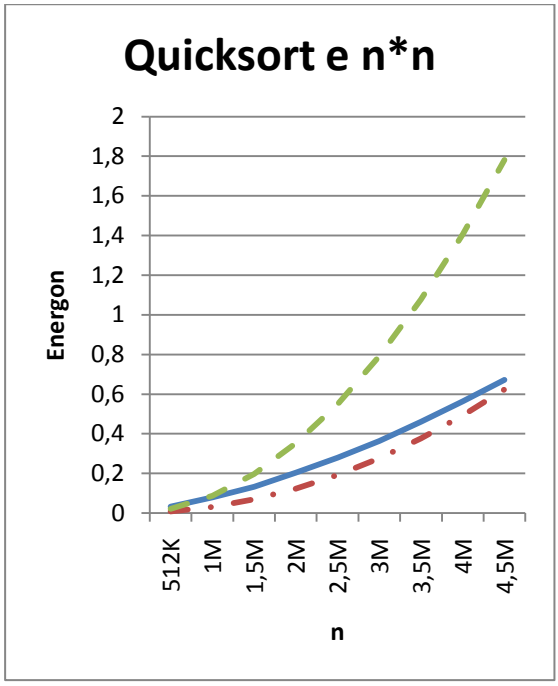

Figura 24: Quicksort e n*n

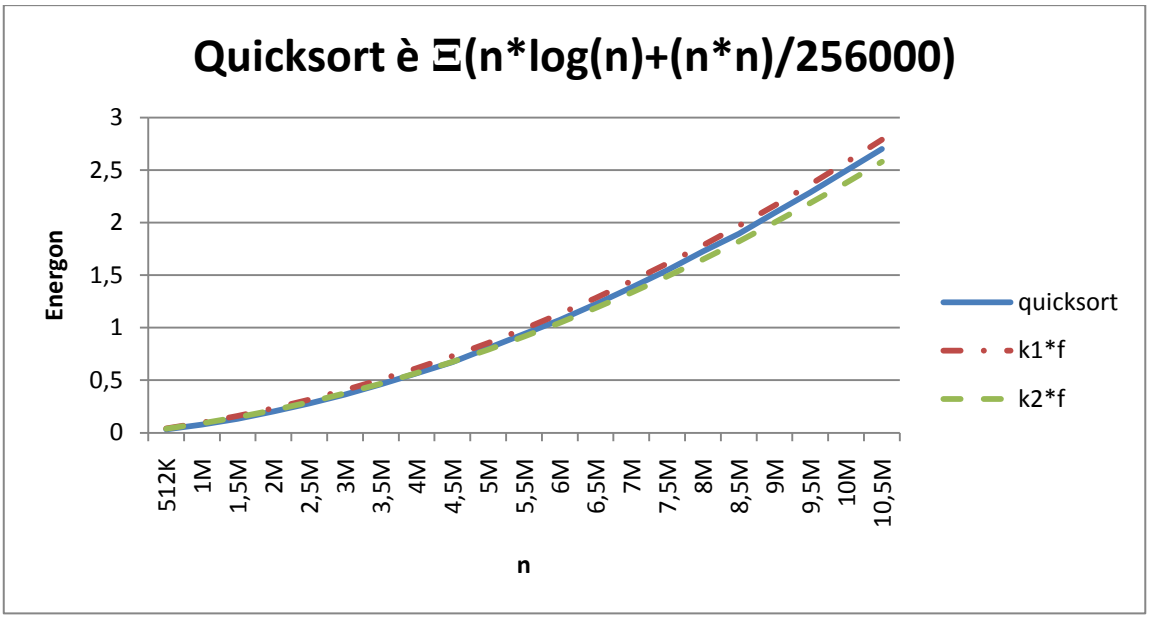

Figura 25: Quicksort

Considerazioni

Pattern di accesso in memoria

È noto come il costo di accesso in memoria influisca profondamente sulle performance temporali degli algoritmi perché attualmente le memorie sono gerarchiche e constistono di molti livelli (L1, L2, DRAM, HD, SSD, etc.) ognuna dotata di diversa capacità, banda e latenza. L'accesso alla memoria ai livelli vicini alla CPU è più veloce rispetto a quella ai livelli più lontani di diversi ordini di grandezza. In questa sezione esploreremo questo aspetto dal punto di vista del consumo energetico.

Abbiamo creato un algoritmo che, presi tre parametri a, b ed n, accede ad un array $I[1,n]$ in blocchi di dimensione b, saltando a blocchi per ogni blocco letto, assicurandosi di accedere sempre all'interno dell'array calcolando l'indice di accesso sempre in modulo n. I parametri n e b sono potenze di 2, ed a è $2^k + 1$ in modo da essere sicuri di accedere a tutte le posizioni dell'array. È chiaro che variando i parametri a e b si possono esplorare varie problematiche quali: accesso in memoria sequenziale contro accesso casuale; accesso ad una singola posizione contro accesso ad un blocco di posizioni contigue.

In Figura 26 sono mostrati i primi otto accessi effettuati dall'algoritmo per $n = 16$, $a = 1$, $b = 1$. Per valori piccoli di a si effettua accesso sequenziale all'array.

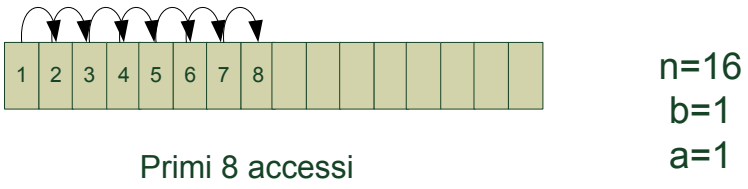

Figura 26: primi 8 accessi per n=16, a=1, b=1

In Figura 27 sono mostrati i primi otto accessi effettuati dall'algoritmo per $n = 16$, $a = 3$, $b = 2$. Al crescere di a l'accesso all'array diventa sempre più *casuale*,

annullando eventuali benefici del prefetching effettuato ai vari livelli delle gerarchie di memoria. Per lo stesso motivo le performance miglioreranno al crescere di b, essendo l'array acceduto a blocchi.

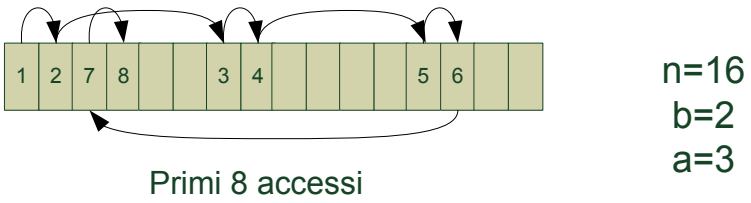

Figura 27: primi 8 accessi per n=16, a=3, b=2

Figura 28 mostra il consumo energetico dell'algoritmo al variare di a e b per $n = 512k$. Come atteso possiamo osservare come al crescere di b il consumo energetico diminuisca, mostrando come un accesso a blocchi ai dati permetta di abbattere fortemente i consumi. In secondo luogo notiamo come all'aumentare di a, e quindi all'aumentare della casualità dell'accesso in memoria, aumenti anche l'energia assorbita. Questo effetto è tanto più presenta quanto aumentiamo la dimensione dell'array.

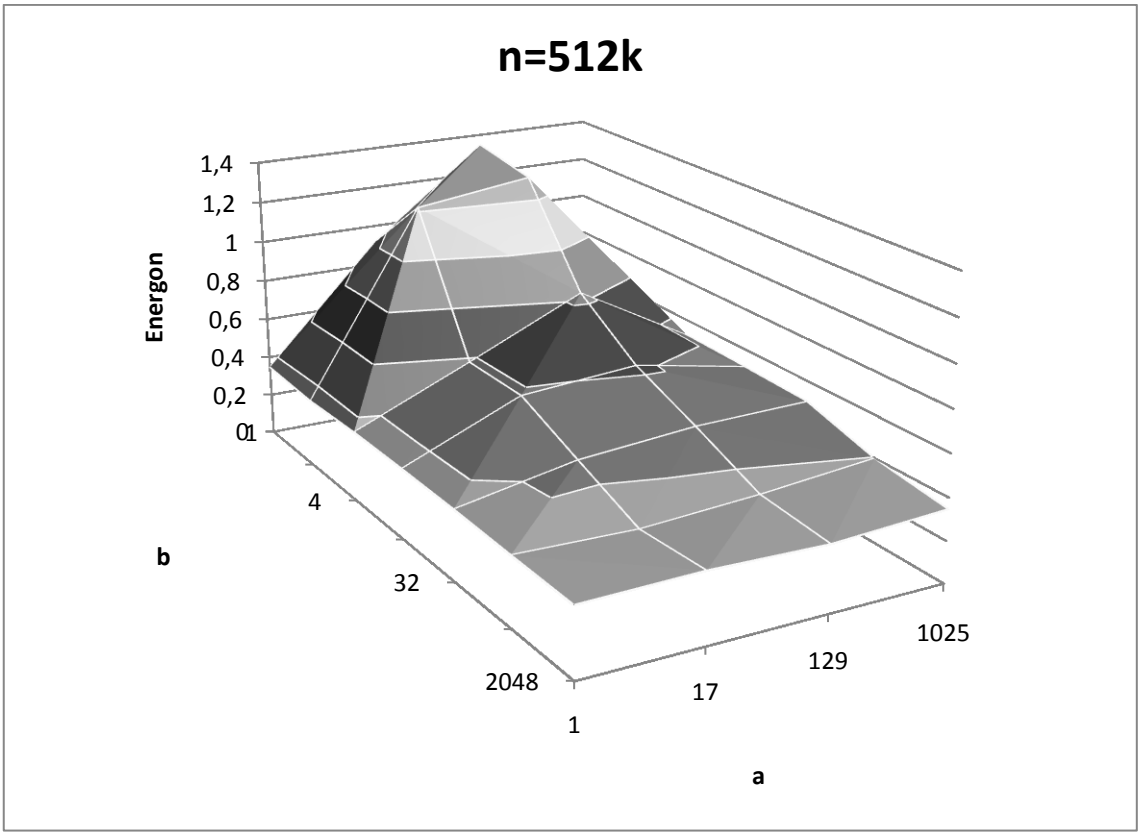

Figura 28: consumo energetico al variare di a e b

Figura 29 mostra la variazione del consumo energetico al crescere di n. Sono stati scelti valori di n, in funzione delle caratteristiche del sistema sul quale sono stati eseguite le misurazioni, in modo che: in un caso l'intero array potesse risiedere in cache primaria (L1, nel nostro caso pari a 64K); nel secondo caso l'intero array potesse risiedere in cache secondaria (L2, nel nostro caso pari a 512K) ma non in cache primaria (L1); nel terzo caso l'array non potesse risiedere in cache secondaria (L2).

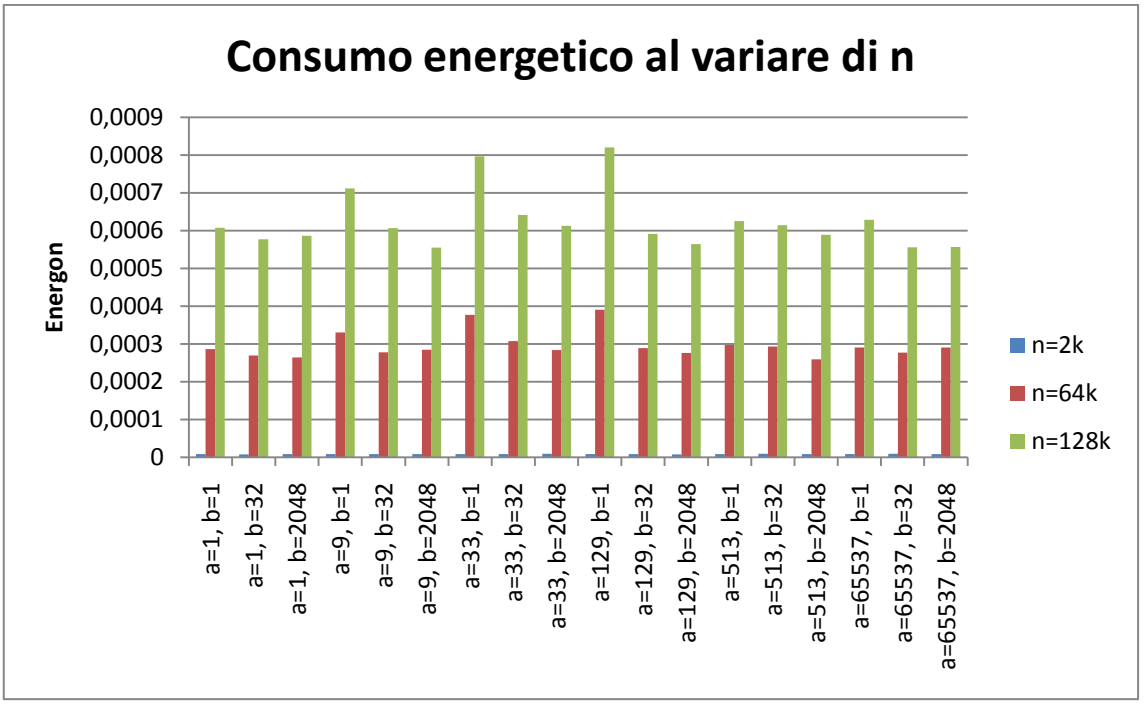

Figura 29: Variazione del consumo energetico al variare di n

È interessante notare di quanti ordini di grandezza aumenti il consumo energetico al salire delle gerarchie di memoria. Da notare anche come il consumo energetico decresca per valori di a molto grandi, effetto dovuto alla diminuizione dell'effetto *random* nell'accesso ai dati, procedendo per salti molto grandi in modulo n infatti diventa probabile cadere in una zona dell'array precedentemente caricata dal prefetching. Figura 30 mostra come il costo di accesso sia sostanzialmente costante al variare di a e b quando i dati sono completamente contenuti in L1.

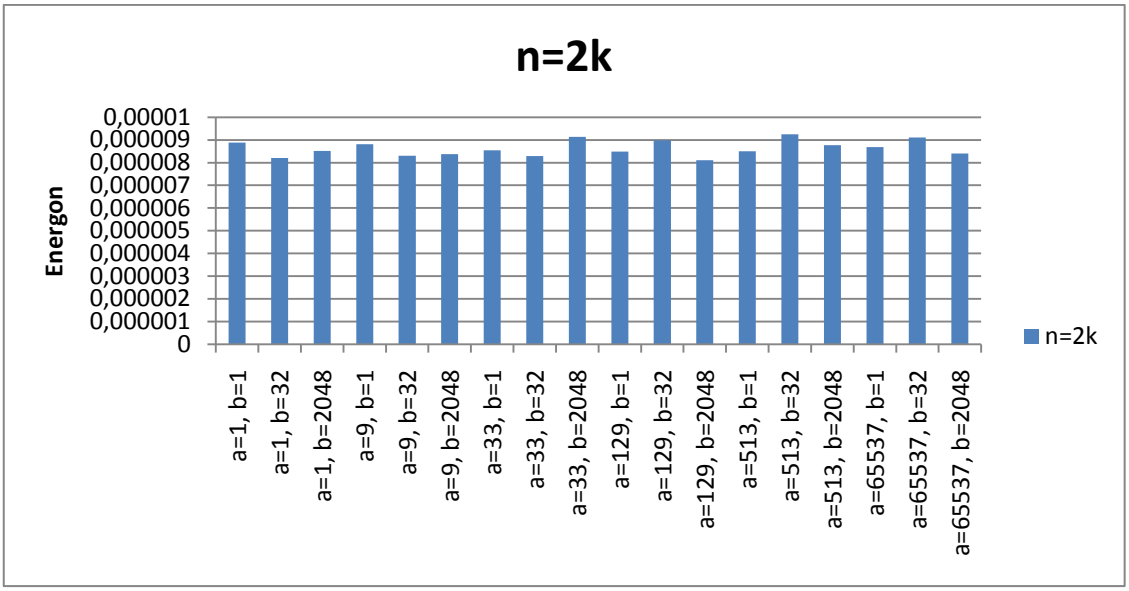

Figura 30: Costo accesso in memoria L1

Le considerazioni fatte fin qui possono essere applicate anche al solo tempo di completamento, Figura 31 tuttavia mostra il rapporto tra l'energia assorbita ed il tempo di completamento per diverse dimensioni dell'array. Possiamo notare come il rapporto cresca insieme alla grandezza dell'array, quindi Energon prende in considerazione aspetti del costo dell'accesso in memoria che non viene reso esplicito dal tempo di completamento a sé stante. Questo grafico mostra inoltre come due algoritmi possano avere stesso tempo di completamento ma diverso assorbimento energetico, in base al modo in cui vengono processati i dati e dalla quantità dei dati acceduti.

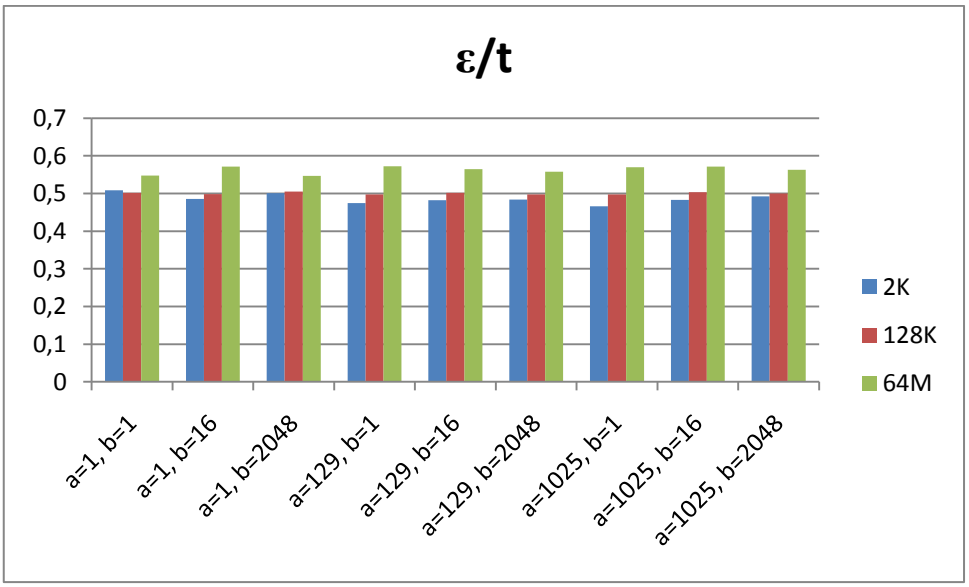

Figura 31: rapporto tra Energon e tempo di completamento

Crediamo che i nostri risultati portino argomentazioni valide all'affermazione in [9]: "*the relation between time- and Energy-efficiency is much intricate*".

Qualità del codice

L'algoritmo descritto nella sezione precedente è stato scritto in tre diversi modi per misurare l'impatto di cicli annidiati e dell'uso della FPU dal punto di vista energetico.

Una prima versione dell'algoritmo (chiamata DL, visibile in Figura 32) è stata scritta usando due cicli annidiati e facendo uso di operazioni di moltiplicazione e divisione in virgola mobile.

```
long pos = 0;
for (long i = 0; i < n / b; i++)
{
    pos = (pos + a) % (n / b);
    for (long j = pos * b; j < (pos + 1) * b; j++)
    {
        sum += Array[j];
    }
}
```

Figura 32: DL, con ciclo annidiato e uso di FPU

Una seconda versione dello stesso algoritmo (chiamata DLint, visibile in Figura 33) fa sempre uso del ciclo annidiato ma evita l'uso della FPU.

```
long sum = 0;
long n_div_b = n/b;
long b_index = (int) log2((int)b);
long pos = 0;
for (long i = 0; i < n_div_b; i++)
{
    pos = (pos + a) % (n_div_b);
    for (long j = pos << b_index; j < (pos + 1) << b_index; j++)
    {
        sum += Array[j];
    }
}
return sum;
```

Figura 33: DLint, ciclo annidiato evitando la FPU

Infine una terza versione fa uso di un solo ciclo senza usare la FPU.

```
long pos = 0;
long delta = (a-1) *b + 1;
for (long i = 0, j=0; i < n; i++)
{
   sum += Array[pos];
   j++;
   pos = (pos + ((j<b)? 1: delta) ) & MASKN;
   j = j & MASKB;
}
```

Figura 34: SL, ciclo singolo senza FPU

Le tre versioni dello stesso algoritmo sono state compilate senza ottimizzazioni e misurate sugli stessi valori di n, a e di b. Figura 35 mostra le misurazioni effettuate sulle tre versioni dello stesso algoritmo rendendo evidente quanto sia influente lo stile di scrittura del codice sul consumo energetico complessivo.

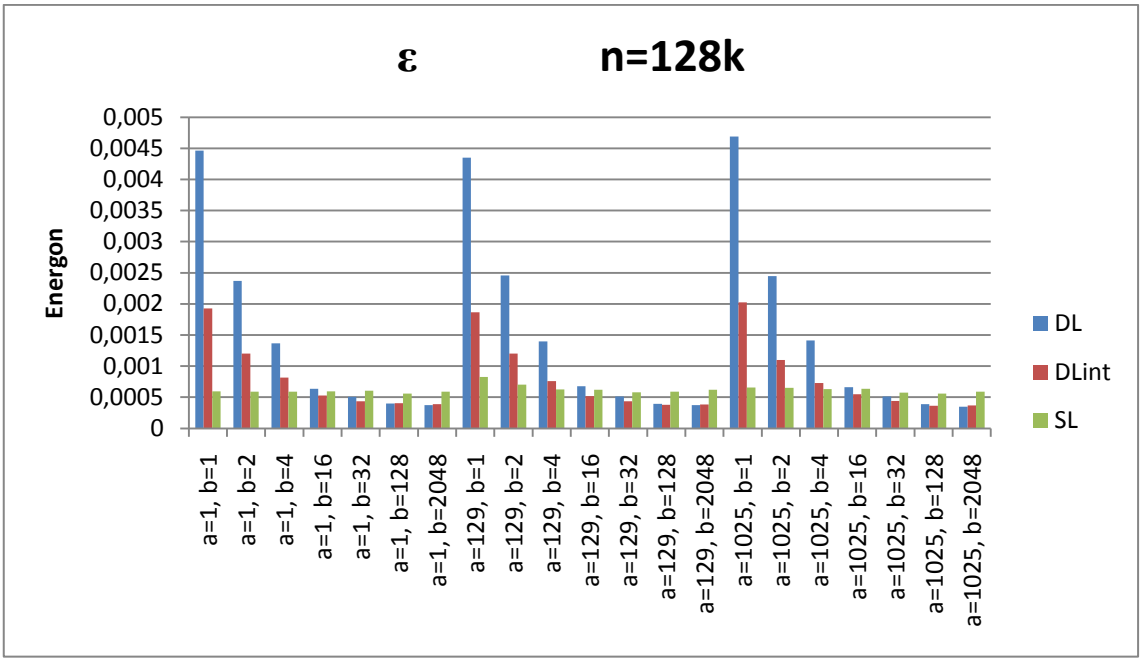

Figura 35: Consumo energetico cambiando lo stile di scrittura del codice

L'uso della FPU influenza pesantemente il consumo energetico, infatti DLint, specialmente per piccoli valori di b che determinano una più frequente esecuzione delle operazioni in FPU, ha un costo inferiore rispetto a DL.

Si noti inoltre come per piccoli valori di b le versioni basate su ciclo annidiato (DL e DLint) siano caratterizzate da un costo energetico di un ordine di grandezza superiore rispetto alla versione SL, dovuto al continuo svuotarsi della pipeline (dell'architettura superscalare usata) provocato dai salti. Per grandi valori di b invece SL ha un costo leggermente maggiore di DL e di DLint perché SL è caratterizzata da un ciclo più lungo laddove invece le versioni con ciclo annidiato minimizzano il codice nel ciclo interno.

Come nella sezione precedente analizziamo ora il rapporto tra l'energia consumata ed il tempo di completamento. Figura 36 mostra come questo rapporto non sia costante al variare di a, b e dello stile di scrittura del codice ma subisca forti oscillazioni di difficile interpretazione. È molto interessante notare come DL abbia generalmente un miglior rapporto Energon/tempo (vedi anche Figura 37).

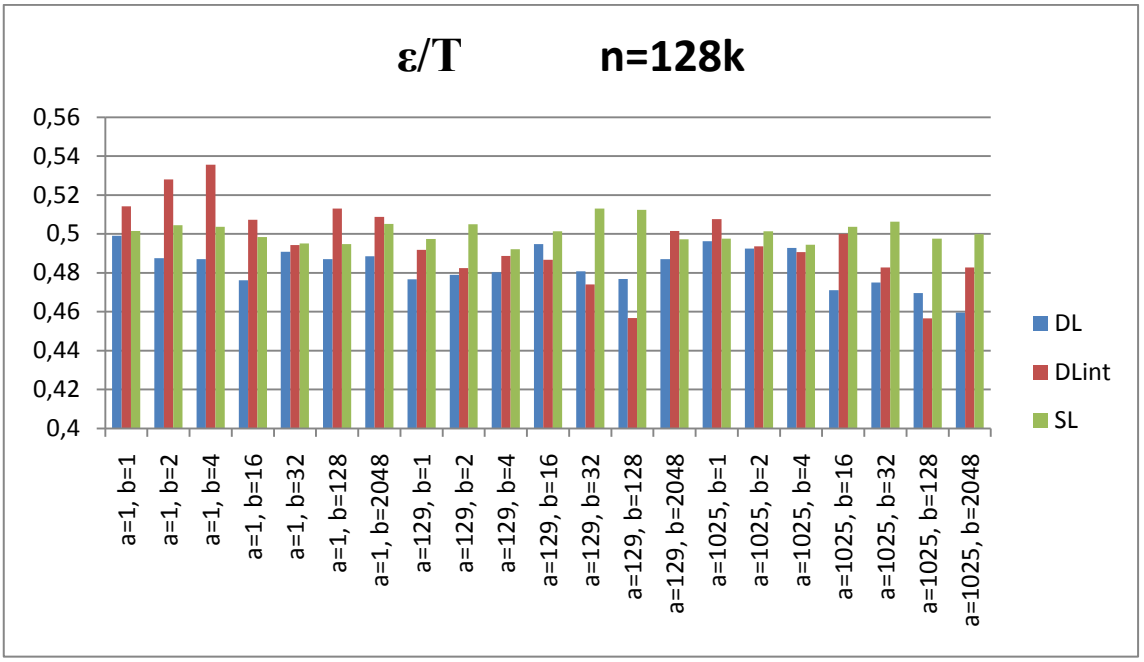

Figura 36: Rapporto tra Energon e tempo di completamento

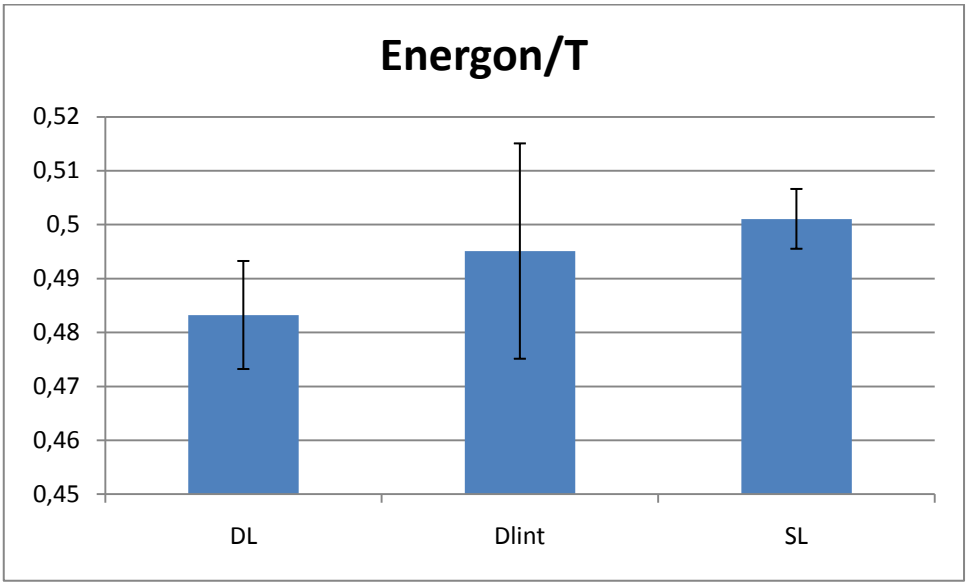

Figura 37: Valori medi e deviazione standard del rapporto tra Energon e tempo di completamento

Multicore

Fino ad ora abbiamo discusso misure energetiche di algoritmi sequenziali, confrontando i nostri risultati con il modello RAM. Tuttavia le caratteristiche delle CPU moderne (quali esecuzione superscalare e presenza di molteplici core) modificano l'efficienza energetica dei sistemi. In letteratura mancano modelli teorici relativi all'efficienza

energetica rispetto all'uso della parallelizzazione. Nella sezione precedente abbiamo visto come l'esecuzione superscalare influisca nel consumo energetico, in questa sezione analizzeremo l'uso di più core.

Idealmente, se ogni Joule fosse utile al calcolo, l'energia E_a richiesta per l'esecuzione di un algoritmo su un dato input dovrebbe essere costante indipendentemente dal numero p di entità di computazione usate. Questo modello è analogo a quello ideale nel quale il tempo di completamento $T_C = \frac{T_1}{p}$, dove T_1è il tempo di completamento dell'algoritmo eseguito su una singola unità di calcolo. La legge di Amdahl [35] corregge questo modello affermando che nel tempo di completamento devono essere prese in considerazione anche le parti intrinsecamente sequenziali del calcolo:

$$T_c(p) = T_S + \frac{T_1}{p}$$

Considerazioni analoghe possono essere fatte per il consumo energetico:

$$W_C(p) = W_O + W_p p$$

Dove W_O è la potenza media assorbita dall'infrastruttura del sistema (il framework di comunicazione, le gerarchie di memoria, le periferiche che collegano i core o i processori nel sistema); e W_p è la potenza media richiesta da ogni core per effettuare i calcoli rispetto allo stato idle. Ottenendo la seguente formula:

$$E_C(p) = T_C(p)W_C(p) = \left(T_S + \frac{T_1}{p}\right)\left(W_O + W_p p\right)$$

Nel caso ideale sia temporalmente che energeticamente abbiamo

$$W_O = 0, \qquad T_S = 0, \qquad E_a = T_1 W_p$$

Definiamo due architetture astratte di riferimento, chiamate CC e IC, ambedue CPU multicore con p cores. L'architettura CC (core-cost) ha $W_O = 0$ (in questo modo l'infrastruttura non ha alcun costo); l'architettura IC (infrastructure cost) ha $W_p = 0$ (in questo i core non hanno costo).

CC è una architettura ideale dal punto di vista energetico, viene usata energia solo dalle unità di calcolo. Nel caso ideale temporalmente l'algoritmo è privo di parte sequenziale ($T_S = 0$), quindi completamente parallelizzabile e vediamo come il comportamento sia quello atteso intuitivamente: il consumo energetico non cambia al variare del numero di unità di elaborazione utilizzate (Figura 38). Nel caso realistico invece in cui la componente inerentemente sequenziale sia presente ($T_S > 0$) il consumo cresce col numero di cores usati (Figura 39).

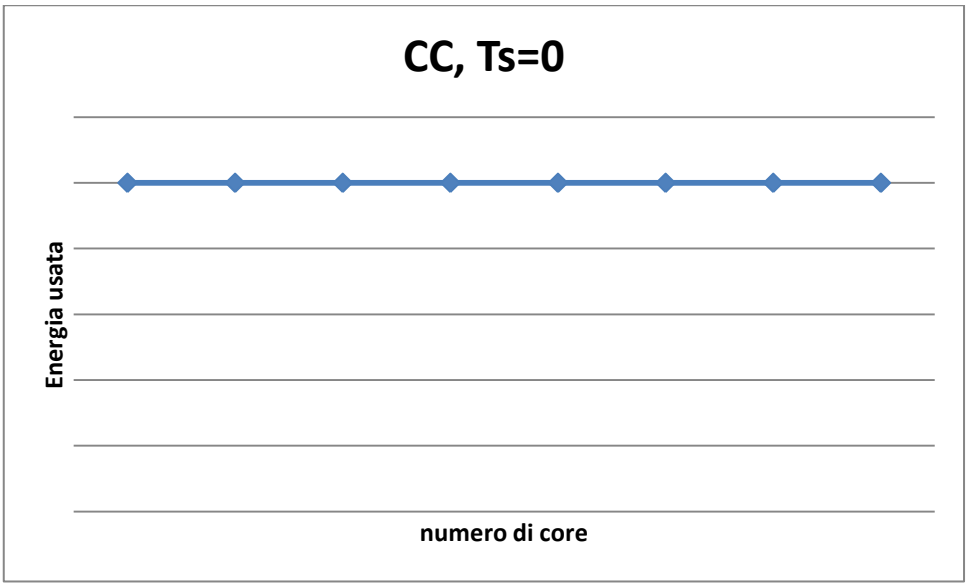

Figura 38: CC nel caso ideale temporalmente

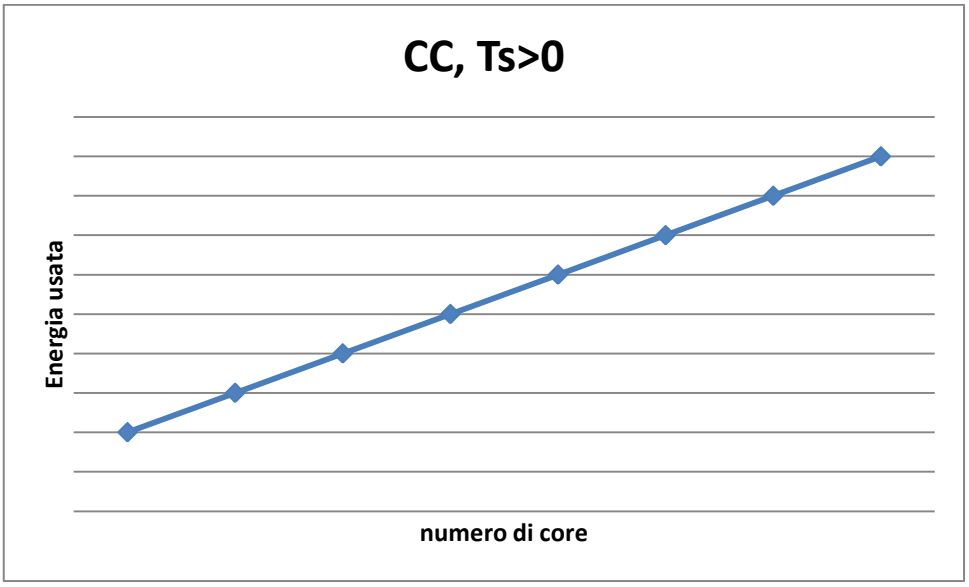

Figura 39: CC nel caso realistico temporalmente

L'architettura astratta IC modella un parallelismo nel quale i core non hanno alcun costo ed esprime quindi l'overhead dell'infrastruttura che diminuirà all'aumentare dei core usati perché a sua volta diminuisce il tempo di completamento. Questo comportamento è presente sia nel caso temporalmente ideale ($T_S = 0$, vedi Figura 40) che in quello temporalmente realistico ($T_S > 0$, vedi Figura 41).

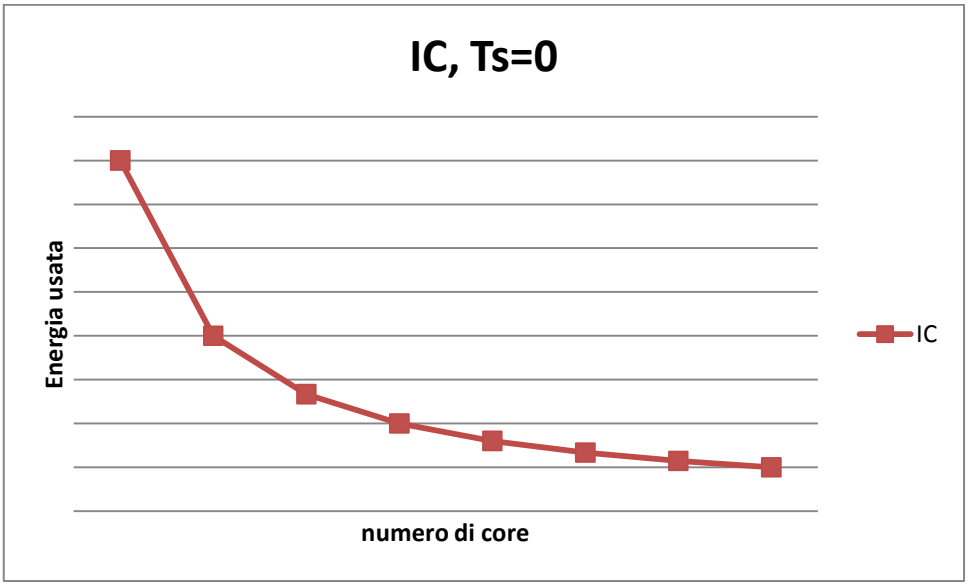

Figura 40: IC nel caso ideale temporalmente

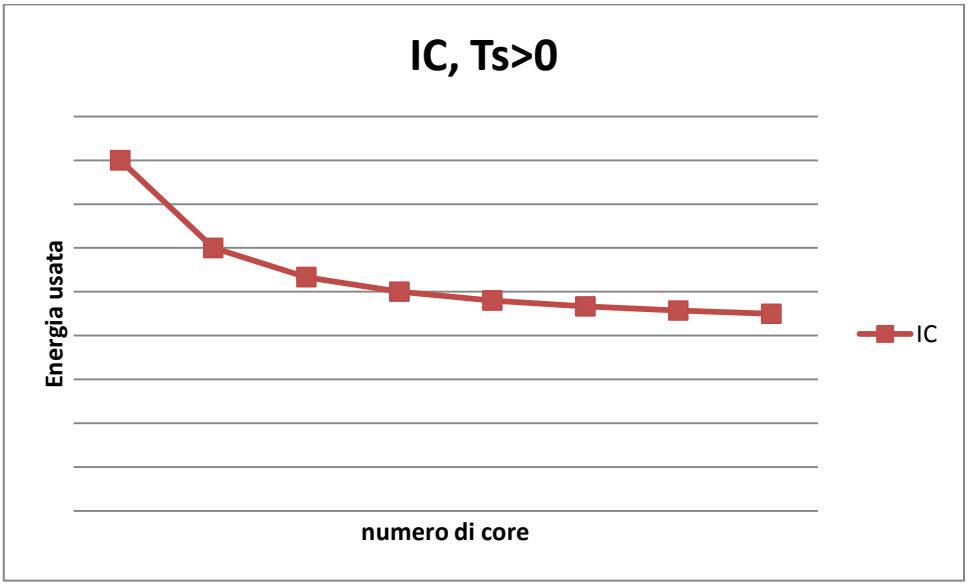

Figura 41: IC nel caso realistico temporalmente

Le architetture astratte CC e IC possono essere usate per modellare il comportamento delle architetture reali: notiamo infatti come il costo energetico complessivo sia ottenuto dalla somma delle due architetture astratte.

Figura 42 mostra l'andamento delle architetture astratte CC e IC, e dell'architettura reale (la somma delle architetture astratte) per $T_S = 0$ (caso ideale temporalmente). Da notare come CC sia fisso al variare del grado di parallelismo.

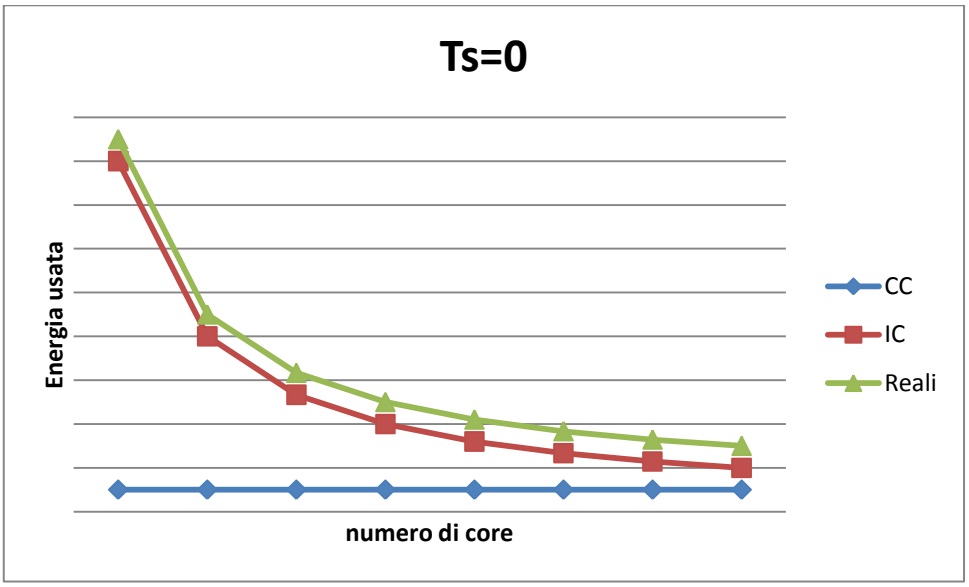

Figura 42: CC, IC e architetture reali, Ts=0

Figura 43 mostra l'andamento delle architetture astratte CC e IC, e dell'architettura reale per $T_S > 0$ (caso reale temporalmente). Da notare come CC cresca al variare del grado di parallelismo, quindi l'architettura reale sarà somma di due termini dei quali uno crescente ed uno decrescente, e potrà quindi verificarsi un minimo. Il punto in cui si verifica il minimo corrisponde al grado di parallelismo che ottimizza il consumo energetico.

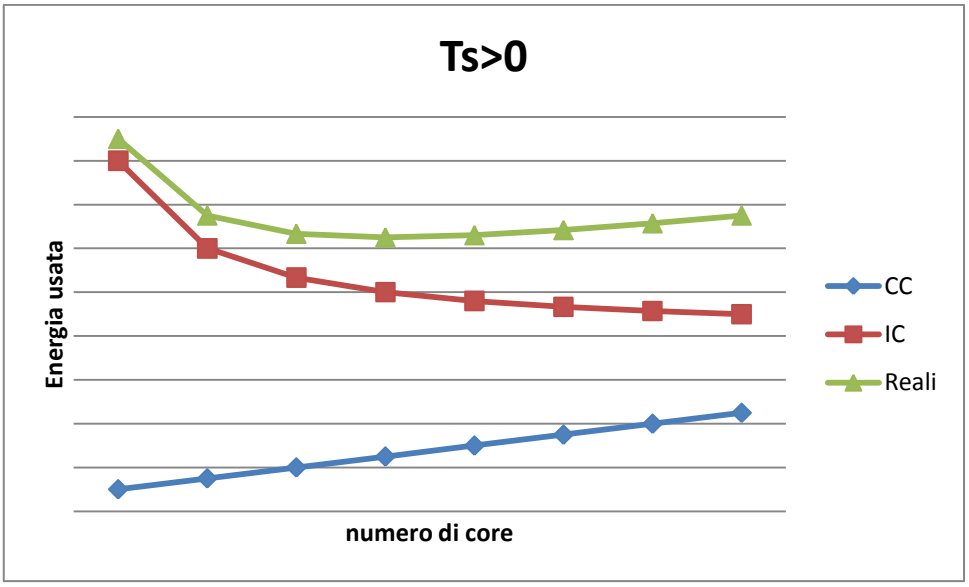

Figura 43: CC, IC e architetture reali, Ts>0

Per verificare la validità del modello appena esposto abbiamo disegnato un semplicissimo algoritmo parallelo e misurato le sue performances temporali (applicando un modello ispirato alla legge di Amdahl) ed energetiche (applicando il nostro modello) al variare del grado di parallelismo.

Abbiamo misurato una semplice scansione lineare su un grande array su un sistema quad-core con hyper-threading, visto dal sistema operativo come un 8 core). L'algoritmo partiziona l'array in p segmenti e li scansiona contemporaneamente, in modo da ottenere uno speedup teorico pari a p. In pratica tuttavia accessi concorrenti alle stesse unità di memoria rallenteranno le prestazioni (lo speedup misurato è 3,81 usando 8 cores). L'effetto è ben modellato dalla legge di Amdahl [35] con un overhead sequenziale pari a $T_S = 3{,}59e-8$ e $T_1 = 1{,}99e-7$, come mostrato in Figura 44. Abbiamo analizzato versioni parallele dell'algoritmo usando $p = 1,2,4,8$ cores.

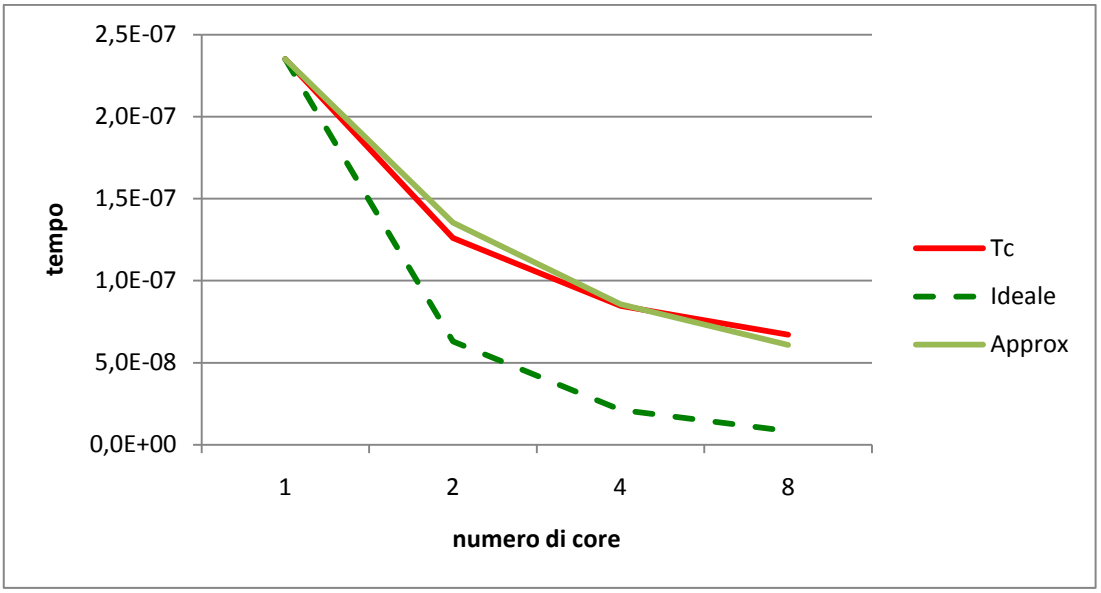

Figura 44: Scansione lineare ideale e tempo reale di completamento rispetto al numero di core usati, e approssimazione usando Amdahl

Figura 45 mostra la potenza media in Watt usata dall'algoritmo usando diversi numeri di core e dimensioni dell'array. All'aumentare del grado di parallelismo aumenta la potenza usata dal sistema ma si riduce il tempo di esecuzione con il risultato di un minore consumo energetico.

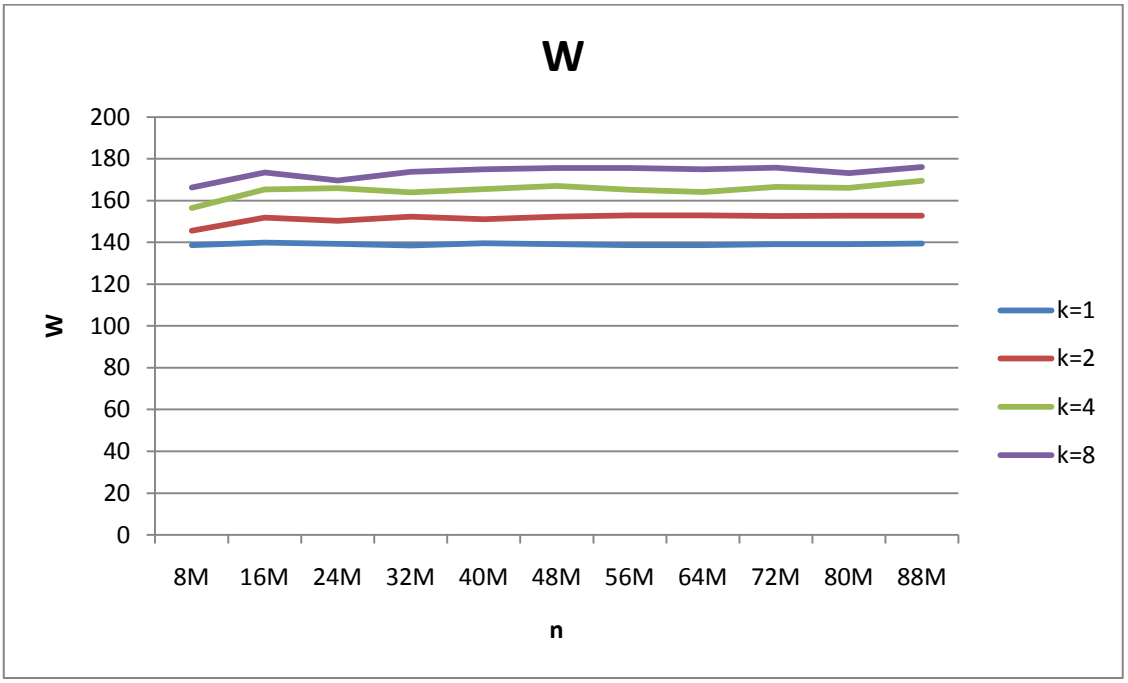

Figura 45: Potenza media usata dagli esperimenti

È stato misurato il consumo energetico della scansione lineare al variare del numero di core usati. Dalle misurazioni è possibile stabilire valori approssimativi di $W_O = 134{,}18W$ e di $W_p = 4{,}92W$ grazie ai quali è possibile disegnare i modelli CC e IC, mostrati in Figura 46.

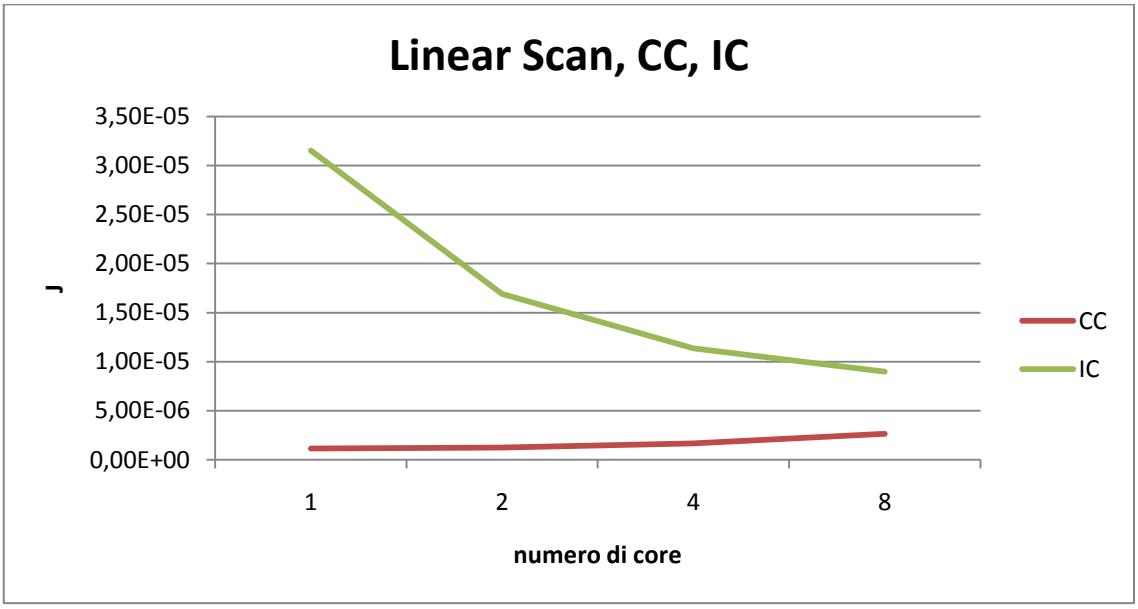

Figura 46: modelli CC e IC calcolati sulle misurazioni

Figura 47 mostra i valori misurati della scansione lineare ed i valori attesi sommando i modelli IC e CC, come si può vedere il modello basato sulle architetture astratte si rivela molto preciso.

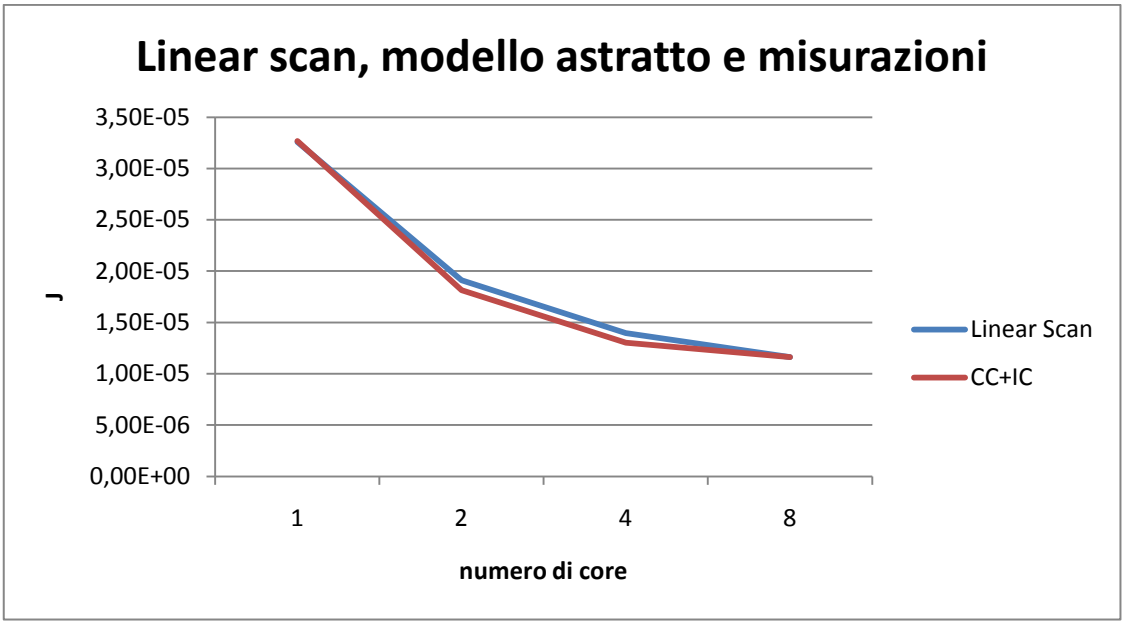

Figura 47: Linear scan, valori attesi e misurazioni

Confrontando CC e IC (in Figura 46) si nota come in questo specifico hardware il costo energetico complessivo dell'infrastruttura sia molto più grande del costo dei core, anche quando il grado di parallelismo è al suo massimo. CC inoltre cresce troppo lentamente per creare un minimo, infatti Figura 47 mostra come si raggiungano i migliori risultati energetici usando tutti i core disponibili.

Sistemi caratterizzati da un più ridotto overhead di infrastruttura, o con un numero maggiore di core, potrebbero mostrare un diverso equilibrio tra CC e IC e potrebbe quindi essere presente un minimo nel grafico del consumo energetico.

Come anche rilevato da [27] la potenza consumata da un sistema multicore aumenta col numero di core attivi in modo non lineare. Figura 48 mostra come l'energia aggiuntiva richiesta per ogni core attivo sia inversamente proporzionale al numero di core attivi, in modo tale che il costo di passare da 1 a 2 core attivi è circa lo stesso di passare da 2 a 4.

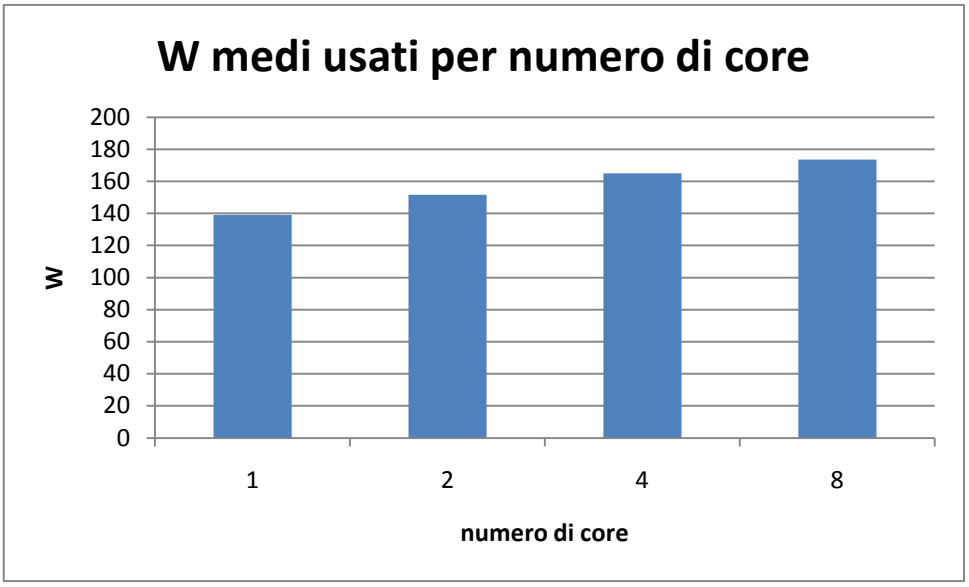

Figura 48: Potenza usata al variare del numero di core attivi

Architetture

Abbiamo misurato due diverse architetture, su sistemi dotati della stessa quantità di RAM (2 Gb): un AMD Athlon 64 X2 Dual Core 4200+ 2,20 GHz; e un Atom N230. Abbiamo eseguito binary-search, merge-sort e quick-sort con il duplice scopo di verificare i risultati raggiunti e precedentemente mostrati, e di mostrare come l'Energon permetta il confronto dei dati tra sistemi diversi.

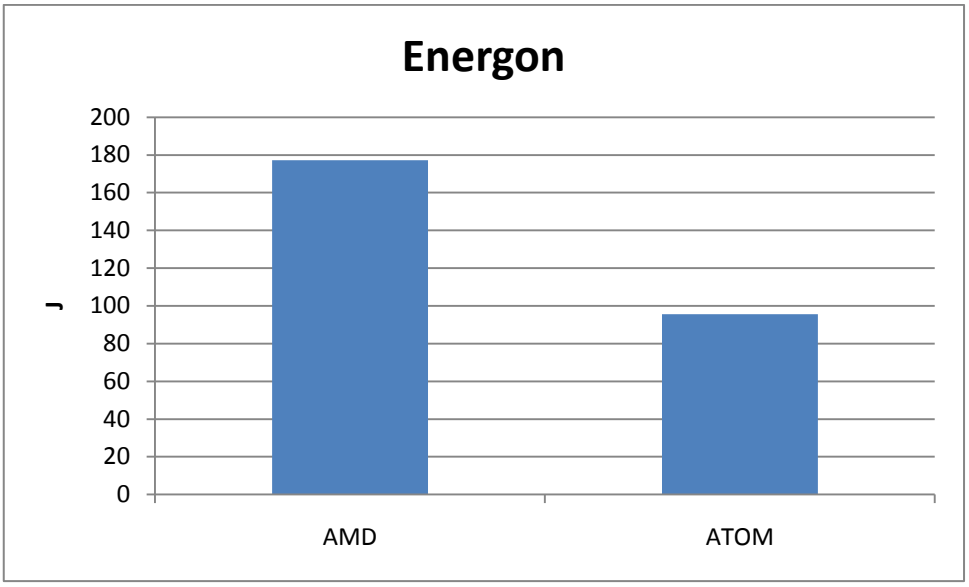

Figura 49: costo del programma Energon su AMD e su ATOM

Per prima cosa abbiamo confrontato il costo del programma Energon eseguito sulle due architetture, per ottenere un'idea generale sull'efficienza generale delle due piattaforme. I risultati confermano la superiore efficienza energetica dell'Intel Atom: assorbe circa la metà dell'energia richiesta dall'AMD. Ovviamente questa stima non include solo il costo della CPU ma dell'intero sistema. Figura 50, Figura 51 e Figura 52 mostrano il consumo energetico del binary-search, i risultati sull'ordinamento mostrano un risparmio ancora maggiore.

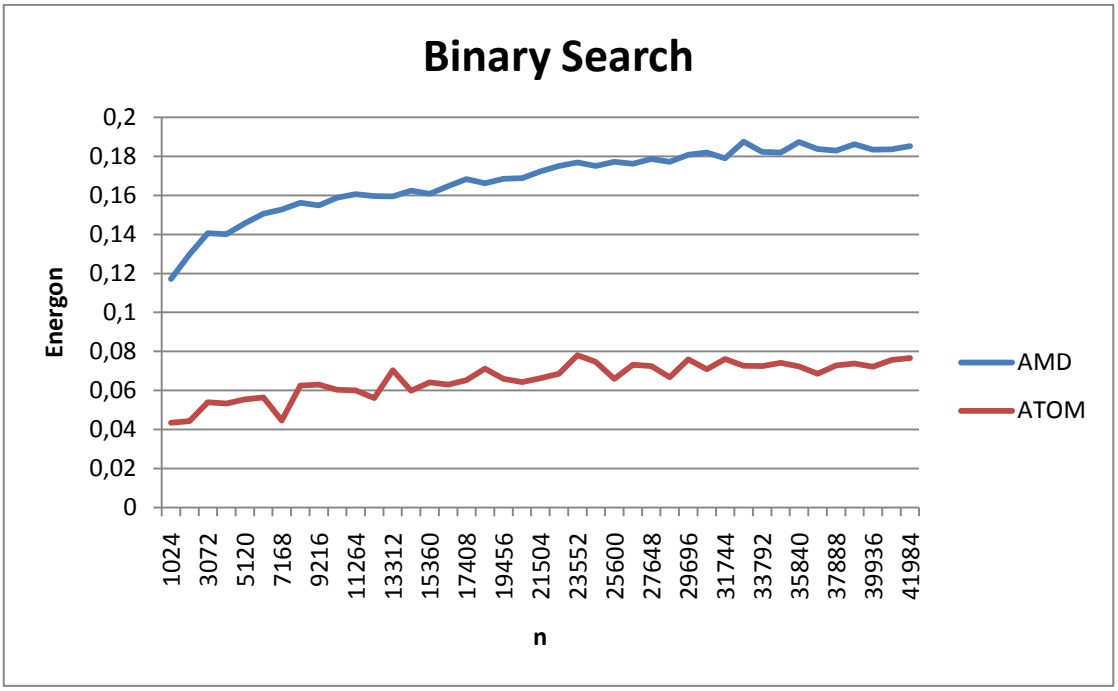

Figura 50: Binary-search, confronto tra AMD e ATOM

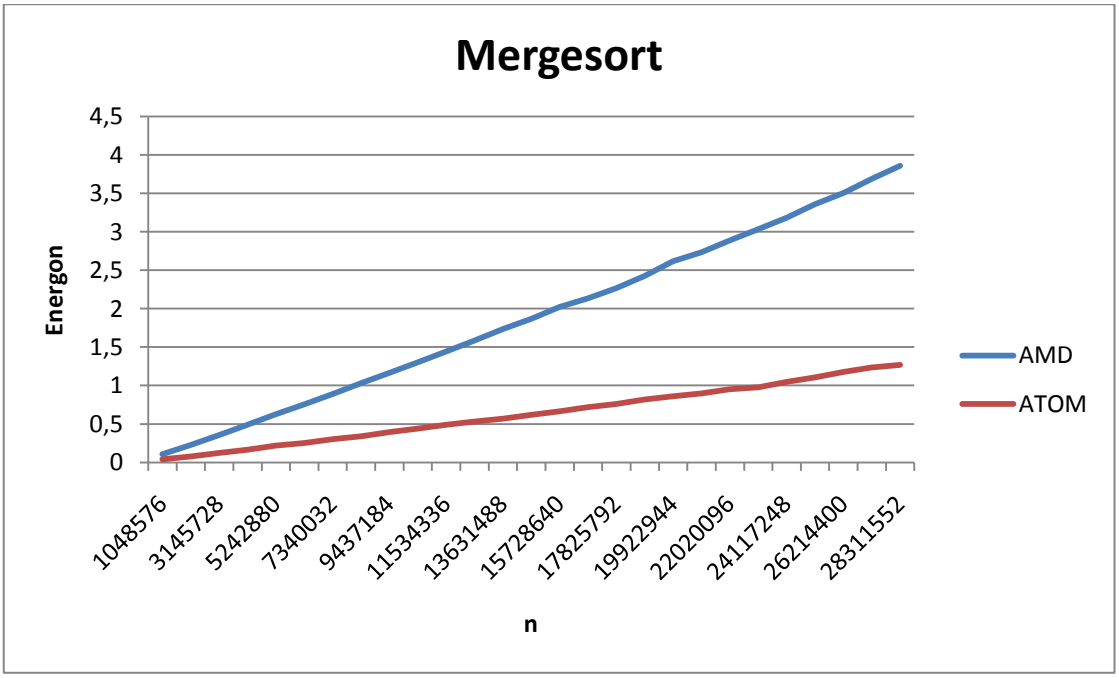

Figura 51: merge-sort, confronto tra AMD e ATOM

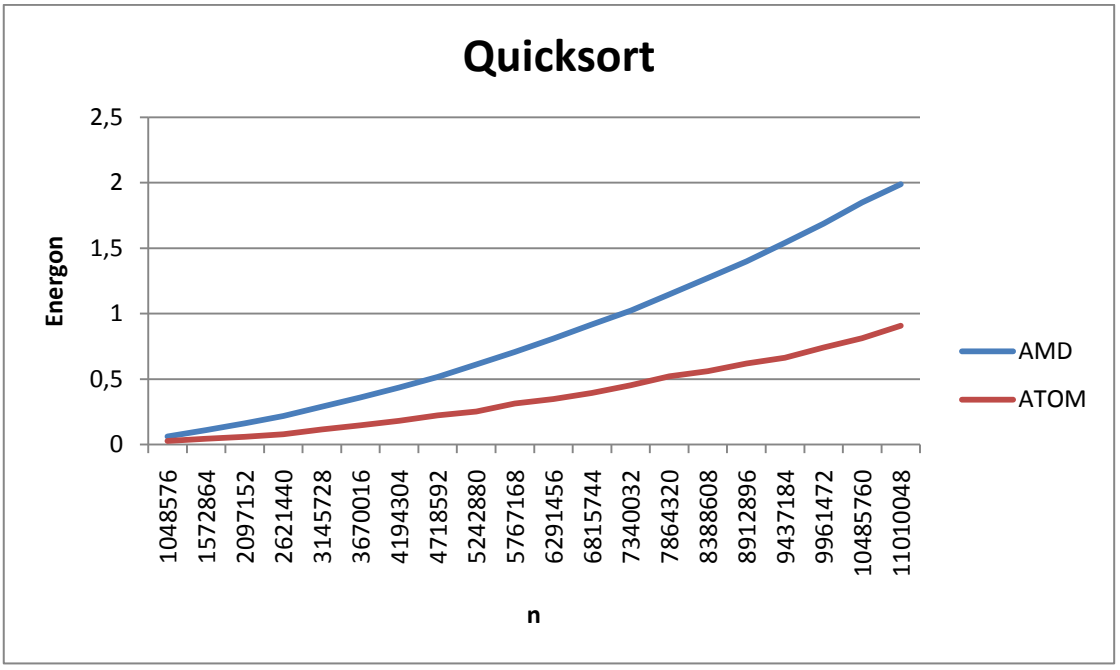

Figura 52: quicksort, confronto tra AMD e ATOM

È interessante notare come i risultati seguano lo stesso andamento sulle due architetture ma non si sovrappongano. Una possibile spiegazione può risiedere nel fatto che il programma Energon attualmente valuta soltanto un solo tipo di istruzione CPU, non prendendo quindi in considerazione altri aspetti delle architetture.

Tuttavia in Figura 53, Figura 54 e Figura 55 possiamo notare come i risultati si sovrappongano perfettamente applicando una costante moltiplicativa k ai risultati per ATOM. Per binary-search è necessario usare una costante di $k = 2{,}5$, per merge-sort $k = 3$ e per quick-sort $k = 2{,}3$.

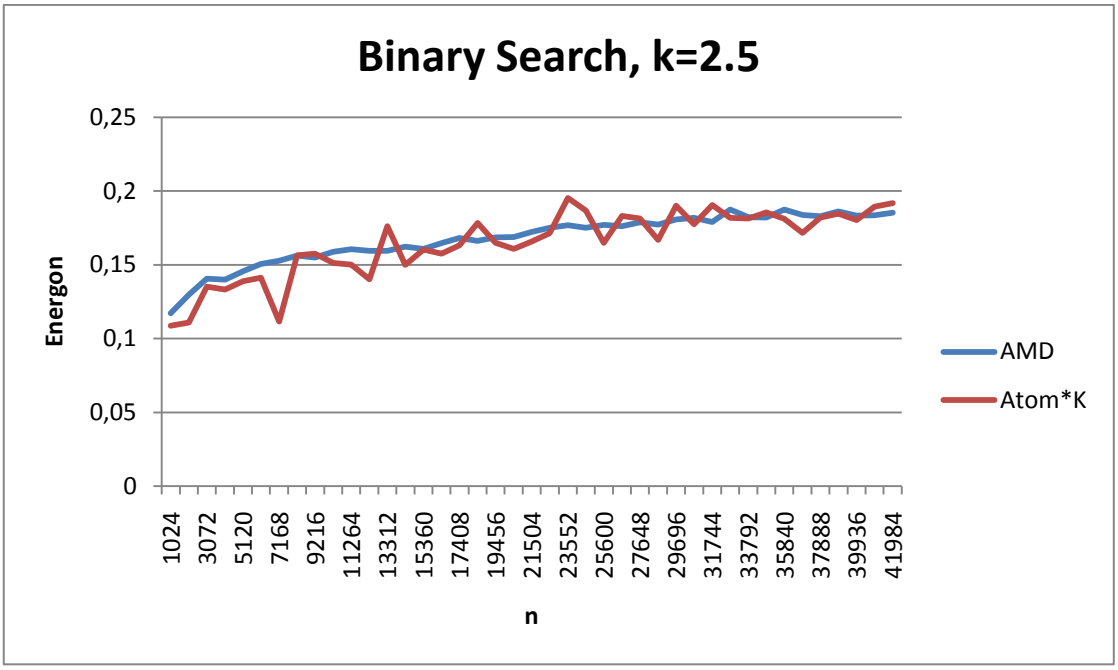

Figura 53: binary-search confronto tra AMD e ATOM applicando k

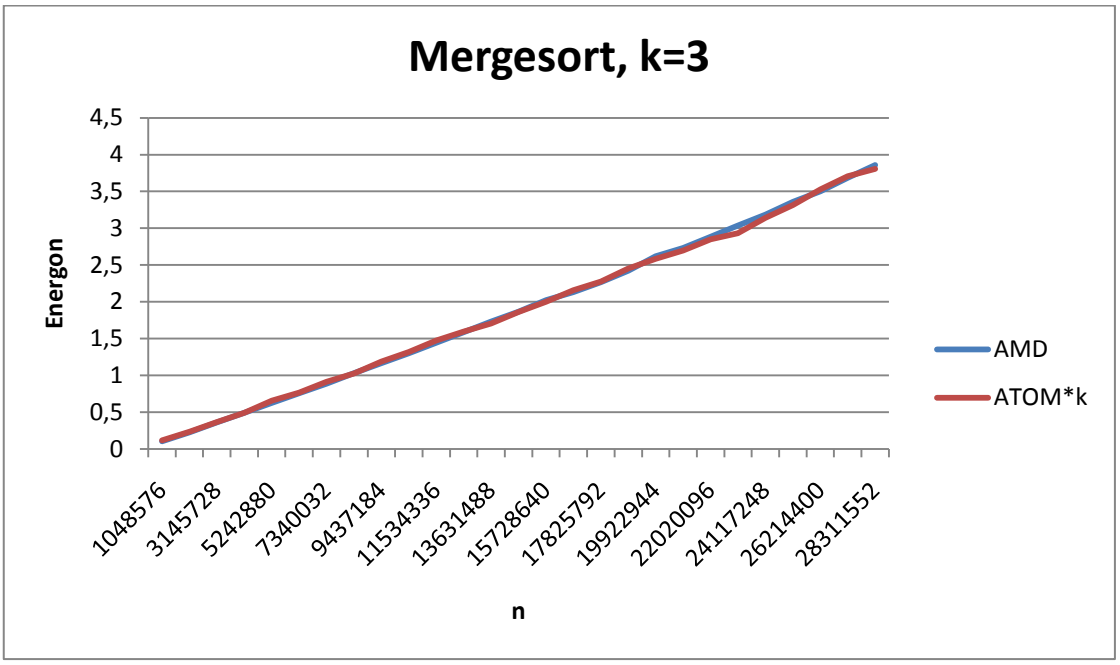

Figura 54: merge-sort, confronto tra AMD e ATOM applicando k

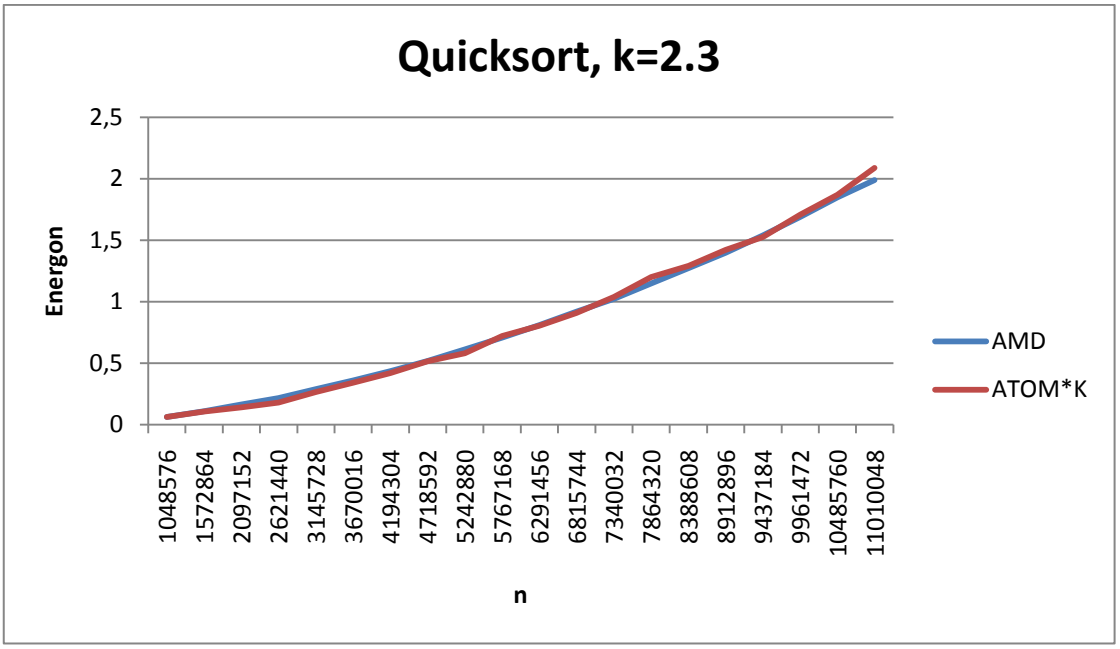

Figura 55: quick-sort, confronto tra AMD e ATOM applicando k

La costante k è simile per tutti gli esperimenti ma non combacia perfettamente. Ci saremmo aspettati la stessa costante k per tutti gli esperimenti, che dipendesse dalle architetture e non dagli algoritmi misurati. Si può notare come quick-sort e binary-search abbiano valori della costante moltiplicativa molti vicini (2,5 e 2,3) e come merge-sort abbia il valore della costante moltiplicativa leggermente distante (3). Riteniamo che questa distribuzione dei valori della costante moltiplicativa sia spiegabile tenendo presente che quick-sort e binary-search condividono un pattern di accesso in memoria *random*, laddove invece merge-sort ha un pattern di accesso in memoria più uniforme.

Da notare come la distribuzione delle costanti moltiplicative neghi una correlazione tra i tempi di completamento: quick-sort e merge-sort hanno simili complessità in tempo ma valori di k diversi, quick-sort e binary-search hanno complessità in tempo molto diverse ma valori simili di k.

Abbiamo riflettuto sul significato che la costante k assume in questo contesto e come cambi il risparmio energetico su un'architettura rispetto ad un'altra al suo variare.

K è definito come:

$$E_A = kE_B$$

dove

$$E_A = \frac{J_A}{\varepsilon_A}, \qquad E_B = \frac{J_B}{\varepsilon_B}$$

in cui J_A e J_B sono il consumo energetico di un certo algoritmo per un certo input su due architetture A e B ed ε_A ed ε_B sono gli Energon che caratterizzano quelle architetture.

Se studiamo il risparmio energetico R_B dato dall'energia in J che viene risparmiata eseguendo l'algoritmo sull'architettura B anziché sull'architettura A possiamo notare come sia ricavabile con semplici sostituzioni algebriche la seguente equazione:

$$R_B = J_A * \left(1 - \frac{1}{k} * \frac{\varepsilon_B}{\varepsilon_A}\right)$$

Essendo J_A, ε_B e $\varepsilon_A$costanti il comportamento di R_B al variare di k seguirà quindi il seguente grafico:

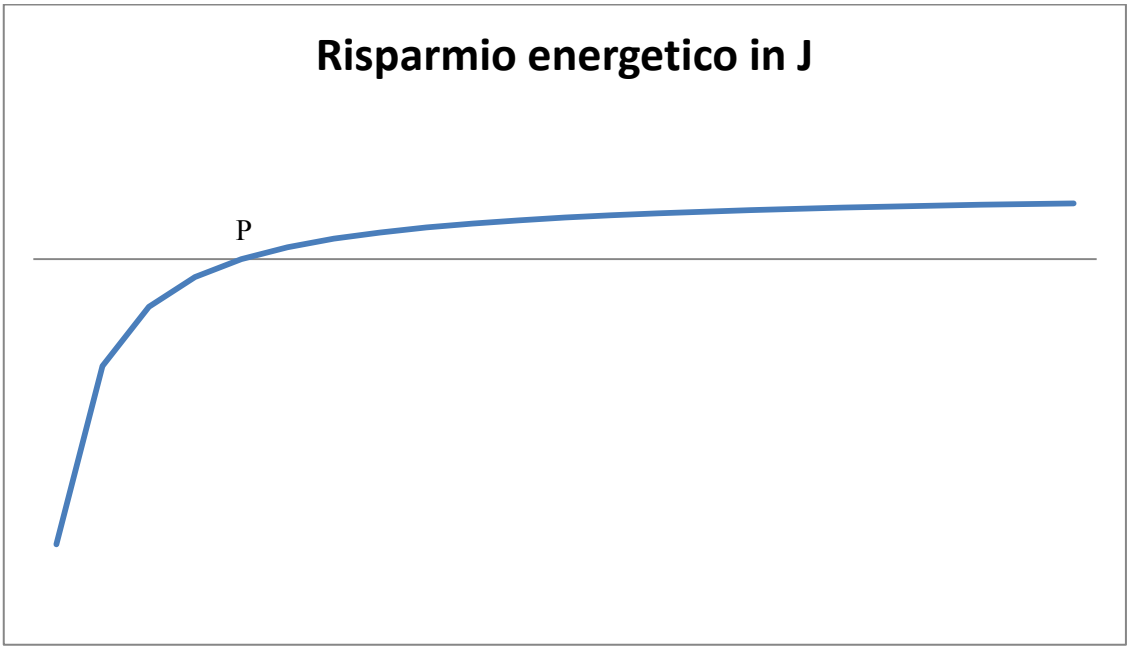

Figura 56: Risparmio energetico dell'architettura B al variare di k

in cui $P = \frac{\varepsilon_B}{\varepsilon_A}$, per valori di $k < \frac{\varepsilon_B}{\varepsilon_A}$ quindi l'architettura A è conveniente rispetto all'architettura B, oltre tale punto abbiamo un risparmio di energia, la funzione tende ovviamente asintoticamente a J_A che è il massimo risparmio energetico possibile.

K è quindi legato allo specifico algoritmo, cambiando algoritmo avremo un diverso valore di k. Una possibile spiegazione del variare di k al variare dell'algoritmo consiste nel fatto che l'Energon da noi usato prende in considerazione solo l'uso intensivo della CPU, senza considerare l'accesso in memoria. Tuttavia al variare dell'architettura il peso dell'accesso in memoria e dell'uso del BUS rispetto al peso della CPU non rimane costante, e questo potrebbe provocare la presenza di k. Sotto questa ipotesi è ovvio che k non sia costante perché non è costante il contributo dell'accesso in memoria al variare dell'algoritmo.

Ci siamo quindi interrogati se non fosse il caso di tentare altre definizioni di Energon in modo da cercare di catturare anche il contributo dell'accesso in memoria e di eliminare quindi k. Tuttavia riteniamo da un lato impossibile eliminare del tutto il fenomeno senza complicare notevolmente il modello, Energon non dovrebbe infatti essere un valore costante ma cambiare in funzione del pattern di accesso in memoria, oppure sarebbe necessario definire un insieme di diversi Energon ognuno disegnato per catturare un aspetto diverso dell'architettura e caratterizzare poi le misurazioni non rispetto ad un unico metro ma rispetto ad un vettore di programmi campione; riteniamo inoltre interessante poter caratterizzare gli algoritmi e le architetture tramite il parametro k, che diventa quindi un ulteriore strumento per esprimerne il costo energetico.

Questo studio preliminare mostra quindi alcuni interessanti spunti degni di ulteriore approfondimento: definire una diversa unità di misura, definire un insieme di unità di misura e studiarne le relazioni, effettuare test sui vari algoritmi su diverse architetture per stabilire i diversi valori di Energon e di k.

Tuttavia quello che ci interessava dimostrare è che la nostra semplice metodologia è solida anche confrontando i risultati ottenuti su architetture diverse, lasciando emergere l'esistenza di k che mostra come il rapporto tra i costi energetici delle varie componenti di un sistema di elaborazione cambi al variare dell'architettura.

Conclusioni

Anche se in alcuni casi abbiamo mostrato come la relazione tra tempo di completamento e consumo energetico possa essere complessa, le due quantità sono strettamente legate tra loro. Scrivere software efficiente dal punto di vista della complessità in tempo generalmente porta anche ad ottenere buona efficienza energetica. Questo è importante perché il tempo di completamento sta perdendo importanza per applicazioni ordinarie, come testimoniato dal sempre maggior numero di programmi scritti in codice interpretato come Python. Dal punto di vista dell'utente un tempo di completamento di un decimo di secondo o di mezzo secondo non fa alcuna differenza, e questo spiega il successo ottenuto da questi ambienti di programmazione. Tuttavia Python è più di cinquanta volte più lento del C in media, vedi [36]. Ci aspettiamo pertanto un consumo più alto dei programmi Python rispetto ai loro equivalenti scritti in C. Gli stessi argomenti valgono per le macchine virtuali, quali Java o Mono (l'implementazione .NET per Linux), rispetto a codice scritto in C. Un'analisi dei costi energetici nascosti del codice di scripting è una delle possibili direzioni in cui sviluppare questa ricerca.

In conclusione, una infrastruttura tecnologica energeticamente efficiente non è ottenuta soltanto attraverso una scelta accurata dell'hardware ma anche attraverso una scrittura consapevole di software. Come testimoniato infatti dal nostro lavoro l'impatto sul consumo energetico del software può essere significativo quanto quello dell'hardware (ed in alcuni casi anche più rilevante), come osservato in [9] "*Algorithmics offers benefits that extend far beyond TCS into the design of systems*".

In nessuna delle proposte presenti in letteratura si trova una metodologia che ponga le basi per la creazione di una teoria del consumo energetico del software astraendosi dall'hardware sul quale viene eseguito, condizione essenziale per permettere la creazione di una disciplina solida, i cui risultati siano espressi in forma tale da poter durare più a lungo della famiglia e modello di calcolatore su cui i test sono eseguiti. Nei numerosi articoli pubblicati inoltre non si trova mai una spinta verso la creazione di una teoria del consumo energetico, non ne vengono cercate le proprietà, generalmente

limitandosi a dare per scontata una aderenza totale alla teoria della complessità computazionale in tempo.

La metodologia di analisi energetica del software presentata in questo lavoro può essere adottata per quantificare in modo semplice ma robusto i benefici di un design energeticamente consapevole del software. In particolare questa metodologia ci ha fornito suggerimenti quantitativi sull'impatto energetico delle computazioni multi-core, dei pattern di accesso in memoria e dello stile di scrittura del codice che possono influire anche per ordini di grandezza. I programmatori dovrebbero pertanto tenere presente questi risultati durante la scrittura del codice.

Le direzioni in cui questo studio potrebbe proseguire sono molteplici: effettuando misurazioni di altri algoritmi, categorizzando i risultati per architettura, in modo da poter confrontare le loro performance energetiche; indagando sulle costanti moltiplicative rilevate nella sezione "Architetture", tentando altre definizioni di Energon, cercando così di assorbire i contributi degli altri componenti dei sistemi di elaborazione quali rete, disco rigido e delle altre periferiche, oppure di renderli espliciti; proseguendo l'indagine sulle architetture multi-core; effettuando test su linguaggi di scripting (quali Python) o sulle Virtual Machines (Java, .NET).

Appendice

Misurare il costo del sistema di misurazione

In questa sezione calcoleremo il costo del sistema di misurazione (vedi Figura 5). Il software di controllo (e le relative connessioni di rete) è sempre attivo sia durante la misurazione degli algoritmi che durante la misurazione del programma Energon inviando messaggi ad intervalli regolari, quindi è un costo fisso e può essere assimilato agli altri servizi del sistema operativo. C'è però un costo che viene erroneamente attribuito all'algoritmo, per la precisione si tratta di quelle operazioni eseguite dall'algoritmo misurato subito prima e subito dopo l'algoritmo vero e proprio (vedi Figura 8): il messaggio di avvio esecuzione algoritmo (std::getline); il messaggio di fine esecuzione algoritmo (std::cout). Per calcolare questi costi (che sono fissi e non dipendono dall'algoritmo eseguito) abbiamo modificato il programma Energon in modo che accetti un nuovo parametro n che determina il numero di volte che Energon stesso verrà eseguito in un ciclo. Il codice sorgente di questo programma (abbiamo chiamato questo programma nEnergon) è visibile in Figura 57, le misurazioni effettuate su nEnergon sono visibili in Figura 58 sovrapposte al parametro n, ci aspettiamo infatti che le misurazioni espresse in Energon si sovrammettano quasi perfettamente col parametro (per la proprietà di composizionalità presentata nel paragrafo "$\boldsymbol{\xi_{f+g} = \xi_f + \xi_g}$", pagina 29).

```
int _tmain(int argc, _TCHAR* argv[])
{
    int n = 1024*1024*1024;
    if (argc>0)
    {
        n = _ttoi(argv[1]);
    }
    int a=0;
    int i = 0;
    std::string line;
    std::cout << "begin\r\n";
    std::getline(std::cin, line);

    for(i=0; i<n; i++)
    {
     __asm
     {
      mov ecx, 40000000h
      mov eax, 0
loop0:
      inc eax
      loop loop0
      mov a, ebx
     }
    }
    std::cout << "end\r\n";
    return 0;
}
```

Energon

Ciclo Esterno

Figura 57: Codice sorgente del programma nEnergon

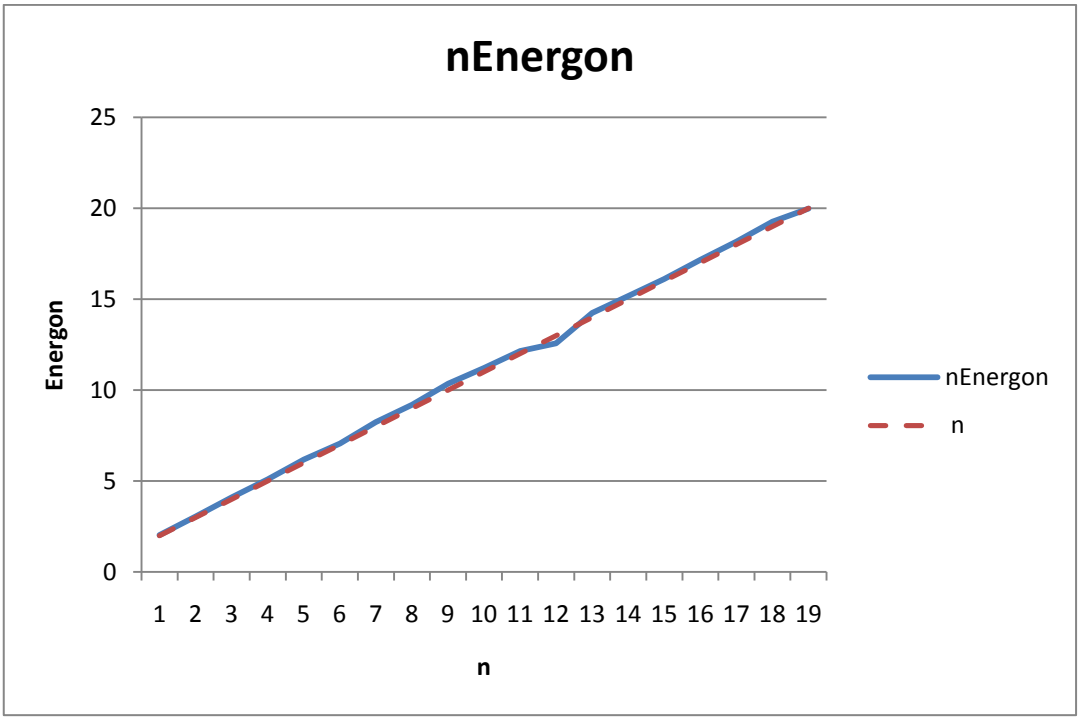

Figura 58: misurazioni del programma nEnergon e confrontate con il parametro n

Possiamo raffinare il modello dei costi del software misurato come

$$J_{tot} = J_{setup} + J_{algorithm}$$

Dove J_{setup} è l'energia assorbita dalle istruzioni dell'ambiente di controllo (std::getline , std::count, networking) e $J_{algorithm}$ è il costo dell'algoritmo vero e proprio. Combinando i risultati di due delle misurazioni di nEnergon possiamo ricavare una stima molto precisa di Energon:

$$J_i = J_{setup} + J_{energon(i)} = J_{setup} + iJ_{energon(1)}$$

$$J_j = J_{setup} + J_{energon(j)} = J_{setup} + jJ_{energon(1)}$$

$$J_{energon(1)} = \frac{J_i - J_j}{i - j}$$

Dove:

- J_i è la misurazione di nEnergon con il parametro $n = i$
- J_j è la misurazione di nEnergon con il parametro $n = j$

- $J_{energon(i)}$ è il costo dell'algoritmo nEnergon *puro*, senza le istruzioni necessarie per comunicare con il software di controllo.
- Per la proprietà di composizionalità sappiamo che $J_{energon(i)} = iJ_{energon(1)}$

Abbiamo applicato questa formula a due delle misurazioni di nEnergon e abbiamo potuto ricavare $J_{energon(1)} = \varepsilon = 310{,}95J$ con una deviazione standard=4,54.

A questo punto siamo in grado di ricavare $J_{setup} = 9{,}04J$.

Questi risultati possono essere usati per raffinare Energon, e per eliminare dalle misurazioni il contributo dei messaggi tra software misurato e software di controllo.

Se andiamo ad esprimere i risultati di nEnergon usando l'Energon appena ottenuto ($310{,}95J$) otteniamo risultati molto precisi. Figura 59 mostra il rapporto tra tali misurazioni ed il parametro n.

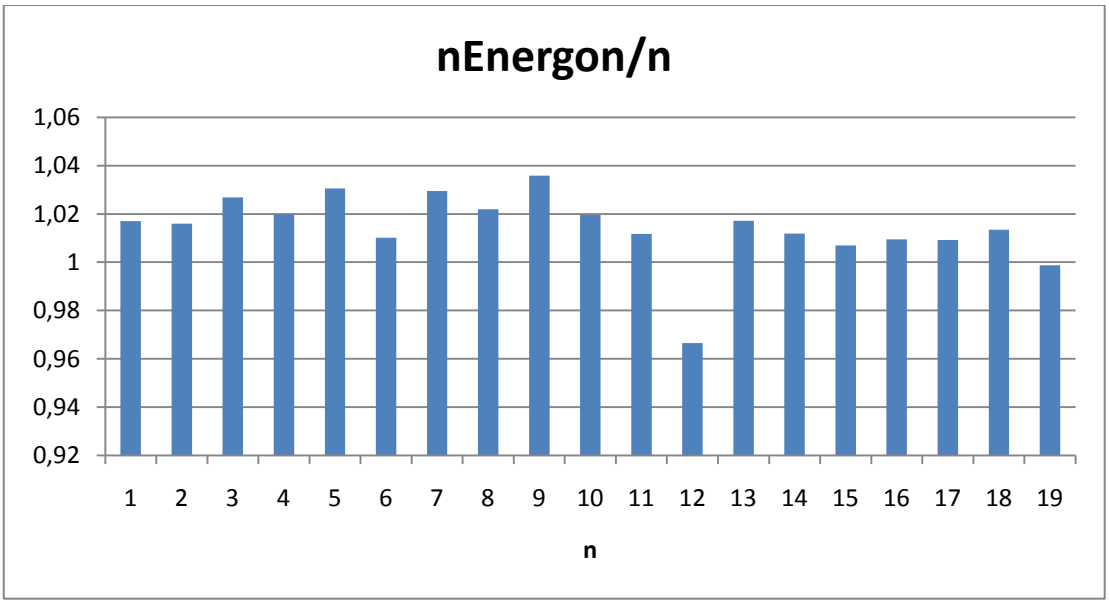

Figura 59: Il rapporto tra le misurazioni di nEnergon espresse in Energon raffinato ed il parametro n

Il valore medio del rapporto è 1 con deviazione standard=0,01.

Come già accennato, è possibile quindi raffinare Energon, ed eliminare dalle misurazioni il contributo dei messaggi tra software misurato e software di controllo. Tuttavia, come possiamo apprezzare in Figura 60, sono quantità di ordini di grandezza diversi e riteniamo quindi che per i nostri scopi sia sufficiente la prima definizione di Energon, più semplice e veloce da misurare.

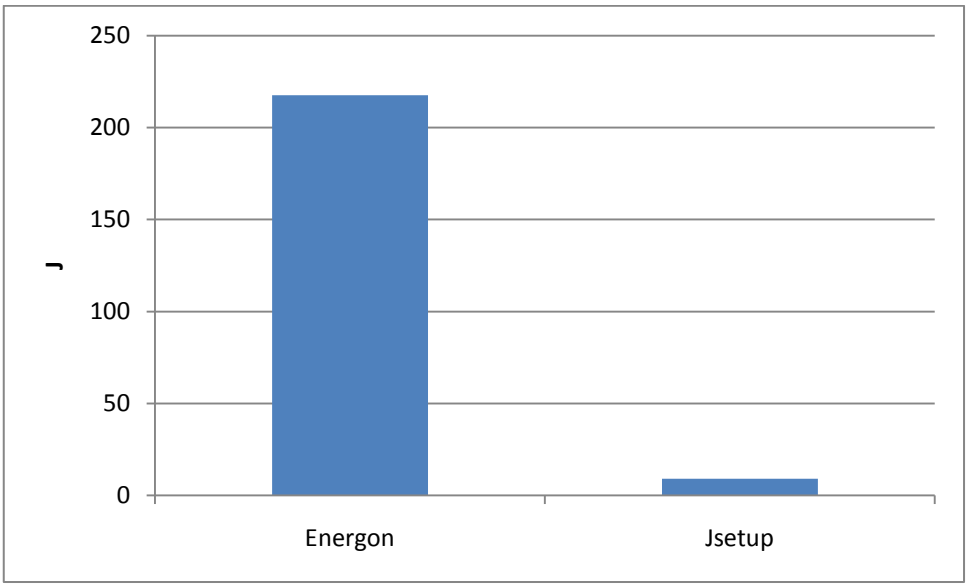

Figura 60: Energon raffinato e Jsetup

Bibliografia

[1] E. Fredkin and T. Toffoli, "Conservative logic," *INTERNATIONAL JOURNAL OF THEORETICAL PHYSICS*, vol. 21, no. 3-4, 1982.

[2] J.T. Russell and M.F. Jacome, *Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors*.: IEEE ICCD, 1998.

[3] "Workshop on the Science of Power Management," 2009.

[4] D., Rivoire, S., et al. Economou, "Full-System power analysis and modeling for server environments.," *Workshop on Modeling, Benchmarking, and Simulation*, 2006.

[5] C. Seo, G. Edwards, D. Popescu, S. Malek, and N. Medvidovic, "A framework for estimating the energy consumption induced by a distributed system's architectural style.," *ACM International workshop on Specification and verification of component-based systems*, 2009.

[6] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations.," in *Annual International Symposium on Computer Architecture (ISCA)*, 2000.

[7] S. Gurumurthi, A. Sivasubramaniam, M.J. Irwin, N. Vijaykrishnan, and M. Kandemir, "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach.," in *International Symposium on High Performance Computer Architecture (HPCA-8)*, 2002.

[8] S. Lafond and J. Lilius, "An Energy Consumption Model for Java Virtual Machine. TR 597.," 2004.

[9] K. Kant, "Toward a science of power management," *IEEE Computer, 42(9)*, 2009.

[10] J.L. Hennessy and D.A. Patterson, *Computer Architecture, Fourth Edition: a*

Quantitative Approach.: Morgan Kaufmann Publisher Inc, 2006.

[11] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms. Third edition.*: MIT Press, 2009.

[12] A. Cisternino, M. Coppola, P. Ferragina, and D. Morelli, "Information processing at work: On a theory of experimental algorithm complexity. Technical Report # 10-13.," 2010.

[13] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," in *International Conference on Computer Aided Design*, 1994, pp. 384-390.

[14] V. Tiwari, S. Malik, A. Wolfe, and M. Tien-Chien Lee, "Instruction level power analysis and optimization of software," *Journal of VLSI Signal Processing*, pp. 223-238, 1996.

[15] R. Mehta, R.M. Owens, M.J. Irwin, R. Chen, and D. Ghosh, "Techniques for low energy software," in *International Symposium on Low Power Electronics and Design*, 1997, pp. 72-75.

[16] N. Chang, K. Kim, and H.G. Lee, "Cycle-Accurate Energy Measurement and Characterization With a Case Study of the ARM7TDMI," in *Proceedings of the 2000 international symposium on Low power electronics and design*, 2000.

[17] J. Flinn and M. Satyanarayanan, "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications," in *Second IEEE Workshop on Mobile Computer Systems and Applications*, New Orleans, 1999.

[18] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: a cycle-accurate energy estimation tool," in *Proceedings of the 37th Annual Design Automation Conference*, 2000.

[19] D. Burger and T.M. Austin, "The SimpleScalar tool set, version 2.0," *ACM SIGARCH Computer Architecture News*, pp. 13-25, Giugno 1997.

[20] J. Chen, M. Dubois, and P. Stenstrom, "Integrating complete-system and user-level performance/power simulators: the SimWattch approach," in *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, 2003, pp. 1-10.

[21] S. Gurumurthi et al., "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach," in *Eighth International Symposium on High-Performance Computer Architecture (HPCA'02)*, 2002.

[22] A. Sinha and A.P. Chandrakasan, "JouleTrack: a web based tool for software energy profiling," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 220-225.

[23] S. Lafond and J. Lilius, "An Energy Consumption Model for Java Virtual Machine," 2004.

[24] N. Vijaykrishnan et al., "Energy behavior of java applications from the memory perspective," in *Proceedings of the 2001 Symposium on JavaTM Virtual Machine Research and Technology Symposium - Volume 1*, 2001.

[25] S. Rivoire, "Models and metrics for energy-efficient computer systems," 2008.

[26] A.J. Martin, "Towards an energy complexity of computation," Pasadena, CA 91125, USA, 2001.

[27] A. J. Younge, G. von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers, "Efficient Resource Management for Cloud Computing Environments," in *International Green Computing Conference*, Chicago, 2010.

[28] T. Li and L. Kurian John, "Run-time modeling and estimation of operating system power consumption," *ACM SIGMETRICS Performance Evaluation Review*, pp. 160-171, June 2003.

[29] J. Enos et al., "Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters," in *International Green Computing Conference*,

Chicago, 2010.

[30] Phidgets. Phidgets - Products for USB sensing and Control. [Online]. http://www.phidgets.com/products.php?category=8&product_id=1122

[31] Codeplex. Codeplex. [Online]. http://energon.codeplex.com/

[32] C. Tseng and S. Figueira, "An analysis of the energy efficiency of multi-threading on multi-core machines," in *International Green Computing Conference*, Chicago, 2010.

[33] L.A. Barroso and U. Hölzle, "The Case for Energy-Proportional Computing.," *IEEE Computer*, pp. 33-37, Dec. 2007.

[34] J.V. Castell and M.J. Cmez-Lechn, *In vitro methods in pharmaceutical research*.: Academic Press, 1997.

[35] G.M. Amdahl, "Validity of the single-processor approach to achieving large scale computing capabilities.," in *AFIPS Conference Proceedings*, Atlantic City, N.J., 1967.

[36] debian_website. Debian language shootout. [Online]. http://shootout.alioth.debian.org/u32/benchmark.php?test=all&lang=all

Indice delle figure

Sommario