Controllo dinamico del consumo di potenza in sistemi a microprocessore

ALESSANDRO BOGLIOLO • LUCA BENINI • BRUNO RICCO
DEIS - Università di Bologna - Viale Risorgimento 2, 40136 Bologna - Italia
GIOVANNI DE MICHELI
CSL - Stanford University - Stanford, CA 94305

Il controllo dinamico del compromesso tra prestazioni e consumo di potenza di sistemi elettronici (*dynamic power management*) è uno strumento che può essere efficacemente utilizzato per aumentare la durata delle batterie dei sistemi portatili a microprocessore, rendere graduale il calo di prestazioni quando l'energia di alimentazione è limitata, contenere i consumi energetici e adattare le prestazioni del sistema al contesto applicativo. In questo lavoro vengono affrontati i principali aspetti del controllo dinamico del consumo di potenza a livello di sistema, ovvero: 1) il progetto di sistemi orientati al *power management*, 2) gli standard industriali, 3) i modelli per la rappresentazione dei componenti del sistema, e 4) l'ottimizzazione delle strategie di controllo dinamico.

1. Introduzione

L'IMPORTANZA DEL CONSUMO di potenza per il progetto di sistemi a microprocessore ha due motivazioni principali: la grande diffusione di applicazioni portatili (quali *laptop computers*, *personal digital assistants*, strumenti biomedici, ecc.) e la pesante incidenza del consumo sui costi di produzione e utilizzo di qualsiasi sistema, portatile e non. Se nell'ambito dei sistemi portatili è ben noto che il consumo impone un difficile compromesso tra le prestazioni e le dimensioni (o la durata) delle batterie che li alimentano, crescente attenzione sta riscuotendo anche l'incidenza dei sistemi a microprocessore sui consumi di energia elettrica domestici e industriali [1]. Inoltre, la forte miniaturizzazione dei componenti (non seguita da una corrispondente riduzione del consumo di potenza) rende estremamente problematico lo smaltimento del calore prodotto durante il funzionamento, richiedendo l'integrazione di apparati di raffreddamento il cui costo può essere paragonabile a quello del sistema stesso. Per questi motivi occorre aumentare l'*efficienza energetica* dei sistemi a microprocessore, riducendone il consumo di potenza senza comprometterne le prestazioni.

I sistemi a microprocessore sono spesso composti da elementi di natura molto eterogenea: circuiti integrati digitali (uno o più microprocessori, memorie, ecc.) e analogici (sensori, filtri, ecc.), componenti elettromeccanici (dischi, attuatori, tastiere), componenti ottici (schermi video, pannelli a cristalli liquidi), ecc. Un tipico esempio di sistema a microprocessore con componenti eterogenei è il computer portatile. Il 36% del consumo di potenza di un moderno *laptop computer* è dovuto allo schermo, il 18% all'hard disk, il 18% ad apparati di comunicazione, il 7% a dispositivi con consumi non critici (mouse, tastiera) e solo il 21% a circuiti integrati digitali (tra cui memoria e CPU) [2]. Questi dati indicano che se anche si riducesse di 10 volte il consumo di potenza dei circuiti digitali presenti in un *laptop*, il consumo complessivo risulterebbe ridotto di meno del 20%.

Benchè numerosi accorgimenti vengano efficacemente applicati durante il progetto di ogni componente (elettronico e non) per ridurne il consumo [3, 4, 5, 6], a livello di sistema esistono altri gradi di libertà che possono essere efficacemente sfruttati per ridurre ulteriormente il consumo totale, essenzialmente poichè non tutti i componenti di un sistema sono contemporaneamente coinvolti nell'elaborazione. Molti di essi trascorrono la maggior parte del tempo in uno stato inattivo (*idle*) nel quale potrebbero essere spenti per risparmiare potenza senza compromettere la funzionalità del sistema. Nella pratica, però, spegnere e riaccendere un componente richiede un tempo non trascurabile e comporta un costo sia in termini di consumo che di prestazioni.

Il controllo *dinamico* del consumo di potenza (*dynamic power management*) è una metodologia di progetto che ha lo scopo di minimizzare il consumo senza compromettere le prestazioni, sfruttando le condizioni di inattività. In generale, tecniche di *dynamic power management* possono essere applicate gerarchicamente ai singoli componenti [7, 9]. A livello di sistema, l'implementazione di tali tecniche richiede l'introduzione di un componente (*power manager*) incaricato di osservare l'attività e controllare lo stato dei singoli componenti secondo una strategia che in seguito chiameremo *politica*. L'efficacia del *power management* dipende sia dalla natura del carico di lavoro del sistema che dalla politica adottata per controllarne il consumo.

Il progetto di sistemi a controllo dinamico del consumo di potenza può essere visto come un processo a due fasi: progetto architetturale orientato al controllo di potenza (*design for power-management*), e progetto della strategia di controllo (*policy optimization*). L'obiettivo del *design for power-management* è quello di costruire un sistema sufficientemente flessibile per sfruttare i gradi di libertà necessari al controllo della potenza. Per far questo è necessario definire nuove cifre di merito che guidino sia il partizionamento del sistema e l'allocazione delle risorse (isolare e rendere separatamente controllabili le

parti del sistema temporaneamente inattive), che il progetto dei componenti (che devono essere *power manageable*, cioè in grado di lavorare a diversi livelli, o *stati*, di consumo e prestazioni e di passare dinamicamente dall'uno all'altro). L'obiettivo della fase di policy *optimization* è quello di sfruttare al meglio i gradi di libertà offerti dal sistema e dall'applicazione. La formulazione e la soluzione del problema di *policy optimization* saranno ampiamente discussi in questo lavoro.

Le strategie di *power management* a livello di sistema possono essere realizzate in forma *hardware* [7] o *software* [8]. Nel caso di sistemi particolarmente semplici può essere vantaggioso implementare le strategie in hardware, modellando il *power manager* come una macchina a stati finiti (FSM) che osserva lo stato dei componenti del sistema e ne controlla le modalità di funzionamento. Nel caso di sistemi digitali a microprocessore, è invece possibile affidare al software il controllo degli stati di consumo di potenza. Per sistemi a microprocessore, in particolare, il livello software più adeguato all'implementazione del *power manager* è il sistema operativo (OS). Tale implementazione risulta particolarmente vantaggiosa poichè affida il controllo dinamico del compromesso tra prestazioni e consumo allo stesso *software* (il sistema operativo) cui è tipicamente affidata la gestione di tutte le risorse.

L'implementazione di *dynamic power management* basato su OS (OSPM) è un problema di *hardware-software codesign*, in quanto le risorse *hardware* devono essere interfacciate al *software*, e sia il sistema operativo che le risorse devono essere progettati in modo da cooperare con il *power manager*. Recenti iniziative hanno preso vita in ambito industriale per fornire supporto *hardware e software* al *power management* basato su OS. In particolare, l'iniziativa *OnNow* di Microsoft delinea le specifiche di una nuova generazione di sistemi a microprocessore in grado di integrarsi nella vita di tutti i giorni reagendo in tempo reale a stimoli esterni di diversa natura [38]. La specifica prevede che il controllo dinamico del consumo di potenza sia tra le funzioni di base del sistema operativo. Intel, Microsoft e Toshiba hanno inoltre compiuto uno sforzo congiunto per standardizzare l'interfaccia tra l'OSPM e i componenti *hardware power manageable*. Lo standard proposto è l'*Advanced Configuration and Power Interface* (ACPI) [36].

La diffusione di sistemi conformi ad ACPI e *OnNow* da un lato semplificherà il problema di *co-design*, definendo a priori i ruoli di *hardware* e *software*, dall'altro renderà ancora più critico il problema di *policy optimization*, sia perchè la crescente disponibilità di componenti *power manageable* aumenterà i gradi di libertà a disposizione e la complessità del problema, sia perchè il supporto *hardware* e *software* rimuoverà i limiti implementativi alla complessità delle strategie.

In questo lavoro analizziamo i diversi aspetti del *dynamic power management* a livello di sistema. Nella prossima sezione discuteremo alcuni dei problemi di progetto di sistema orientato al *power management*, con particolare riferimento alle tecniche di individuazione delle condizioni di inattività e ai meccanismi di spegnimento delle risorse inattive. Descriveremo inoltre gli standard *OnNow* e ACPI e i precedenti contributi scientifici sull'argomento: Nella sezione 3 affronteremo il problema di modellistica a livello di sistema. In particolare introdurremo modelli stocastici per descrivere in modo astratto il comportamento del sistema dal punto di vista del compromesso tra prestazioni e consumo, e discuteremo le ipotesi su cui tali modelli si basano. Nella sezione 4 descriveremo diversi approcci per risolvere il problema di ottimizzazione condizionata delle strategie di *power management*.

2. Progetto di sistema orientato al power management

I requisiti di base per l'implementazione di un meccanismo di controllo dinamico del consumo di potenza sono la capacità di osservare l'inattività di un componente del sistema e la disponibilità di adeguati meccanismi per effettuarne lo "spegnimento" in modo rapido ed efficiente. Ovviamente, il controllo dinamico del consumo di potenza è applicabile con successo solo se la rivelazione dell'inattività di un componente può essere effettuata a basso costo (in termini di prestazioni e consumo).

Possiamo classificare l'inattività di una risorsa di sistema in due categorie: *inattività esterna* e *inattività interna*. Il primo tipo di inattività è fortemente correlato con il concetto di *inosservabilità* delle uscite della risorsa: diciamo che una risorsa è esternamente inattiva ogniqualvolta la sua presenza non è necessaria per il corretto funzionamento del sistema, o equivalentemente, la risorsa non è utilizzata. Quando una risorsa è inosservabile, è possibile ridurre il consumo di potenza semplicemente portando la risorsa in uno stato inattivo a basso consumo di potenza (questo processo è detto *shut down* o spegnimento).

Si ha inattività interna quando l'uscita di una risorsa rimane costante e lo stato interno della risorsa non cambia anche se gli ingressi cambiano. In questo caso è la funzionalità stessa della risorsa a determinarne l'inattività. In generale, è possibile individuare condizioni di inattività esterna tramite tecniche di analisi molto più astratte e semplici di quella usate per rivelare inattività interna. Per questa ragione, tecniche per il controllo del consumo di potenza basate su inattività interna sono generalmente applicate in un ambito relativamente ristretto (in particolare, queste tecniche sono state prevalentemente formulate per circuiti elettronici digitali in tecnologia CMOS statica).

Un semplice esempio di inattività esterna è offerto dal sistema di memoria di massa (hard-disk) di un personal computer che resta a lungo inutilizzato quando l'utente sta svolgendo attività che impegnano prevalentemente la CPU. In queste condizioni, il sistema non ha (momentaneamente) bisogno dell'hard disk. Un semplice esempio di inattività interna è quello di un microprocessore che legge una sequenza di *nop*. In questo caso, è la funzionalità del microprocessore stesso a stabilire che l'output e lo stato interno non hanno cambiamenti di rilievo.

Per il resto della sezione ci concentreremo su metodi per sfruttare l'inattività esterna di risorse di sistema al fine di ridurre la dissipazione di potenza. Tali metodi possono essere applicati a sistemi eterogenei descritti ad un alto livello di astrazione. Il lettore interessato a una dettagliata analisi di tecniche per lo sfruttamento dell'inattività interna in

circuiti digitali CMOS è invitato a fare riferimento a [7].
Assumendo che l'inattività possa essere rivelata in modo efficiente ed economico, è importante avere a disposizione metodi adeguati per forzare lo *shut-down* di risorse inattive. Il metodo più comunemente impiegato nel dominio dei circuiti integrati digitali è noto come *clock gating*. In questo schema, il segnale di clock viene disabilitato quando la risorsa diventa inattiva. L'arresto del clock implica una forte riduzione del consumo di potenza: i registri sono disabilitati e non possono caricare nuovi valori, pertanto l'attività interna al circuito viene minimizzata. In tecnologia CMOS, assenza di attività interna implica riduzione quasi completa del consumo di potenza.
Una tecnica di shut-down ancora più radicale (e generale) del clock gating è lo spegnimento dell'alimentazione, che garantisce la totale eliminazione del consumo di potenza ed è applicabile a qualsiasi tipo di risorsa. Sfortunatamente, lo spegnimento della alimentazione comporta diversi svantaggi. In prima istanza, è necessario memorizzare informazioni necessarie per il proseguimento dell'attività in memorie non volatili. Questa operazione può essere costosa sia in consumo di potenza che in tempo necessario per completarla. Inoltre, ripetute accensioni e spegnimenti tendono a diminuire la vita media di molti dispositivi di uso comune nei sistemi elettronici (tipici esempi sono terminali video o hard disks).
In generale, è necessario accettare compromessi tra potenza dissipata da una risorsa in stato di shut-down, e costo in termini di tempo, potenza e/o perdita di affidabilità per compiere la transizione da e verso lo stato di shut-down. Per questa ragione, molte risorse sono progettate per avere multipli stati di shut-down, ognuno dei quali rappresenta un diverso punto di compromesso. Per meglio illustrare questa importante caratteristica, consideriamo i seguenti esempi.

- Moderni microprocessori a basso consumo di potenza [17] possono, in caso di inattività, essere posti in uno o più stati a basso consumo di potenza. Solitamente, viene definito uno stato di *idle* ove il segnale di clock non viene propagato alle varie unità del processore, ma il generatore interno del clock viene mantenuto attivo. Un secondo stato di inattività, detto *sleep*, consente di diminuire ulteriormente il consumo di potenza fermando anche il circuito di generazione del clock. L'uscita dallo stato di *idle*, a seguito di una richiesta di attivazione esterna, è estremamente rapida e richiede in genere pochi cicli del clock di processore. L'uscita dallo stato di *sleep* è invece molto più lenta e richiede spesso tempi dell'ordine delle decine di millisecondi, perchè l'inizializzazione del generatore di clock e la stabilizzazione del suo periodo di oscillazione sono processi lenti.
- Un dispositivo di memoria di massa (hard-disk drive) può avere multipli stati di shut-down. Consideriamo ad esempio due stati. Nel primo, è disabilitata solo l'interfaccia elettronica tra la meccanica del disco e il sistema, mentre il disco continua a ruotare. Nel secondo caso, anche il motore del disco è fermo, e la dissipazione di potenza è minimizzata. Chiaramente, l'uscita dal primo stato può essere estremamente veloce, poichè richiede l'attivazione di soli dispositivi elettronici. Al contrario, l'uscita dal secondo stato è generalmente diversi ordini di grandezza più lenta, perchè il motore deve superare l'inerzia meccanica del disco è portarlo alla velocità di rotazione richiesta per il funzionamento a regime. Inoltre, continue accelerazioni e decelerazioni possono ridurre sensibilmente la vita media dei componenti meccanici del sistema.

Questi semplici esempi illustrano il compromesso (*trade-off*) tra prestazioni, costo, affidabilità e dissipazione di potenza. Il progetto di schemi dinamici per il controllo di potenza richiede attenta considerazione di tutti questi fattori, perciò non è ovviamente possibile risolvere il problema in modo generale con semplici e generiche euristiche, come quella di portare la risorsa nello stato di minimo consumo di potenza non appena essa diventa inattiva. Strategie ottimali di controllo devono ridurre la dissipazione di potenza, minimizzando l'impatto in termini di prestazioni e affidabilità.

2.1. Power management per microprocessori

I processori per sistemi *low-power* implementano diversi schemi avanzati per ridurre il consumo di potenza [16, 17, 18, 21, 31]. Alcuni schemi consentono di rallentare il clock, altri permettono di disabilitare selettivamente ed autonomamente, senza controllo esterno, il clock di alcune unità funzionali, come l'unità di calcolo in virgola mobile *floating-point unit*. Infine, è possibile (come illustrato nell'esempio della sezione precedente) bloccare completamente il microprocessore in uno stato inattivo con basso consumo di potenza. In quest'ultimo caso, il processore può tornare attivo solo a seguito di una interruzione esterna.
In generale, limitarsi a rallentare la velocità del clock senza diminuire contestualmente la tensione di alimentazione non porta ad alcun vantaggio. Infatti, la potenza consumata dal microprocessore è proporzionale alla frequenza di clock f, alla capacità commutata C e al quadrato della tensione di alimentazione V ($P \sim C V^2 f$), ma il tempo t necessario per completare un qualsiasi compito è in prima approssimazione inversamente proporzionale alla frequenza di clock ($t \sim 1/f$). L'energia necessaria per completare un compito è data dal prodotto del consumo di potenza per il tempo di esecuzione ($E = P\ t$), quindi il consumo di energia è invariante con la velocità del clock. La riduzione della frequenza di clock aumenta il tempo medio di risposta senza ridurre l'energia spesa dal microprocessore.
Se è possibile diminuire la tensione di alimentazione concorrentemente con la riduzione della frequenza di clock, si ha una diminuzione quadratica del consumo di energia che a volte può essere sufficiente a giustificare l'aumento del tempo di elaborazione. In alcuni casi, però, come ad esempio durante l'uso interattivo, i tempi di elaborazione devono essere rapidi anche in sistemi low-power. Inoltre, microprocessori che possano cambiare dinamicamente la propria tensione di alimentazione sono stati finora presentati solo come prototipi di ricerca [32] e non sono disponibili sul mercato. Pertanto, la riduzione della frequenza di clock a tensione di alimentazione costante è utilizzata soltanto per ridurre la potenza in sistemi in cui il microprocessore è scarsamente utilizzato anche quando attivo. Quesa situazione può verificarsi quando la performance del sistema è limitata da componenti diversi dalla CPU.

In pratica, l'utilizzo di stati di inattività a basso consumo di potenza ha avuto maggior successo della riduzione della frequenza di clock. Nell'ipotesi che il tempo di accensione e spegnimento sia trascurabile, e altrettanto lo sia la potenza dissipata durante tali tempi, diventa estremamente conveniente forzare il microprocessore in uno stato a basso consumo di potenza non appena esso diventa inattivo. Questa strategia è in generale possibile quando l'ingresso in stato inattivo comporta solamente la disattivazione dei circuiti di distribuzione del clock. In questo caso, il processore può essere riattivato in pochi cicli di clock (un tempo quasi sempre trascurabile, data l'altissima frequenza del clock nei processori attuali) e con minimo costo in potenza.

Vi sono casi, però, in cui il periodo di inattività della CPU è tanto lungo da giustificare l'ingresso del processore in uno stato di inattività profonda (*sleep*) in cui anche il circuito di generazione del clock interno (generalmente un *phase-locked loop*, o PLL) può esere disattivato, e la tensione di alimentazione può essere portata a zero per l'intero chip o per alcune sue sezioni. In tal caso la dissipazione di potenza è virtualmente azzerata, ma il costo in termini di tempo e consumo di potenza per ritornare in stato attivo è radicalmente più alto. Non solo il generatore di clock interno deve essere riattivato, ma anche la memoria interna al processore (registri, cache di primo livello, TLB, ecc.) viente persa durante il periodo di inattività, e può essere necessario doverla trasferire su memoria permanente prima di entrare in stato di *sleep* e recuperarne il contenuto al ritorno in stato attivo. Anche nel caso in cui il salvataggio di dati su memoria permanente non fosse necessario, si avrebbe un costo in prestazioni al ritorno in fase attiva visto che la gerarchia della memoria richiede un tempo non trascurabile per raggiungere uno stato ad alta performance (il tempo necessario per copiare in cache o nei registri le informazioni provenienti da memorie esterne lente).

Molte famiglie di microprocessori a larga diffusione (Pentium, PowerPC, ARM) offrono versioni low-power con istruzioni dedicate per controllare l'ingresso in vari stati di inattività. L'uscita da tali stati è solitamente automatica ed avviene per mezzo di interrupts generati da eventi esterni asincroni, o, in alcuni casi, da timers programmati esplicitamente prima dell'ingresso in stato inattivo. È importante notare che in molti casi il controllo dello stato operativo del microprocessore non può essere lasciato esclusivamente ad una singola applicazione. La vasta maggioranza dei sistemi di elaborazione impiega sistemi operativi per il controllo e la coordinazione delle attività di sistema; al sistema operativo, come vedremo nella prossima sezione, può essere demandato il compito di controllare il consumo di potenza.

2.2. Power management per personal computers

Alcuni standards sono stati recentemente proposti in ambiente industriale per facilitare l'implementazione di schemi avanzati per il controllo dinamico del consumo di potenza. Una prima, semplice specifica di standardizzazione proposta nel 1996 da Intel, Microsoft e Toshiba, è nota come *Advanced Power Management* (APM). La funzione principale dello standard è di definire l'interfaccia tra un algoritmo di controllo (*power management policy*) e l'hardware. La specifica riguarda sistemi di calcolo a microprocessore, a partire da *laptop computers* fino a macchine *desktop* ad alta performance.

In questi sistemi, le funzioni di controllo di alto livello sono espletate dal sistema operativo, mentre il controllo diretto dei componenti hardware è solitamente affidato a software di sistema noto come BIOS (*basic input-output system*). Il BIOS può essere pensato come un livello di astrazione dell'hardware che viene offerto al sistema operativo: in altre parole, il sistema operativo controlla l'hardware attraverso l'interfaccia procedurale offerta dal BIOS. La specifica APM descrive un'interfaccia che deve essere implementata dai progettisti hardware, e i meccanismi di controllo che devono essere implementati dai programmatori del BIOS al fine di facilitare il controllo dinamico del consumo di potenza. È importante notare che APM *non* specifica come implementare l'hardware o come progettare le routines del BIOS, ma si limita a definire in modo univoco un protocollo di comunicazione.

La specifica APM è stata sostituita nel 1997 da un secondo standard, proposto dalle stesse aziende. La nuova specifica è stata chiamata *Advanced Configuration and Power Management Interface* (ACPI). ACPI non è semplicemente una evoluzione di APM, ma presenta alcune carattersitiche peculiari. La più importante innovazione introdotta in ACPI è lo spostamento delle principali funzioni di controllo dal BIOS al sistema operativo. In ACPI, il controllo dinamico del consumo di potenza è affidato direttamente ad un modulo del sistema operativo, che controlla e misura la dissipazione di potenza dell'hardware tramite primitive di sistema simili a quelle correntemente usate per controllarne la funzionalità. ACPI specifica l'interfaccia che deve essere usata dai progettisti di un sistema operativo e dagli sviluppatori di hardware e di *firmware* per consentire un controllo intelligente della dissipazione di potenza del sistema. Come nel caso di APM, ACPI non restringe in alcun modo le funzionalità del sistema operativo o dell'hardware, lasciando la più completa libertà di implementazione.

L'introduzione di ACPI, e il notevole sforzo compiuto da Intel, Microsoft e Toshiba per garantirne la completezza, testimoniano l'avvenuto riconoscimento, da parte delle principali aziende che controllano il mercato dei *personal computers*, delle potenzialità degli schemi di controllo dinamico della dissipazione di potenza. In caso di successo ed adozione generalizzata, ACPI consentirà di semplificare il progetto e l'implementazione di personal computers con basso consumo di potenza e prestazioni non degradate. Microsoft ha ulteriormente incrementato il suo supporto ad ACPI lanciando una iniziativa, chiamata *OnNow*, il cui fine è innalzare il livello di usabilità dei *personal computer*. L'obiettivo di *OnNow* è di più ampio respiro del semplice controllo della potenza dissipata, e può essere riassunto nelle specifiche funzionali che devono essere rispettate da un personal computer.

- Il PC deve essere pronto all'uso non appena l'utente ne effettua l'accensione, similarmente a tutti i più diffusi elettrodomestici.
- Il PC deve apparire spento quando non viene utilizzato, ma deve essere capace di rispondere efficacemente a richieste provenienti in modo non predicibile dall'ambien-

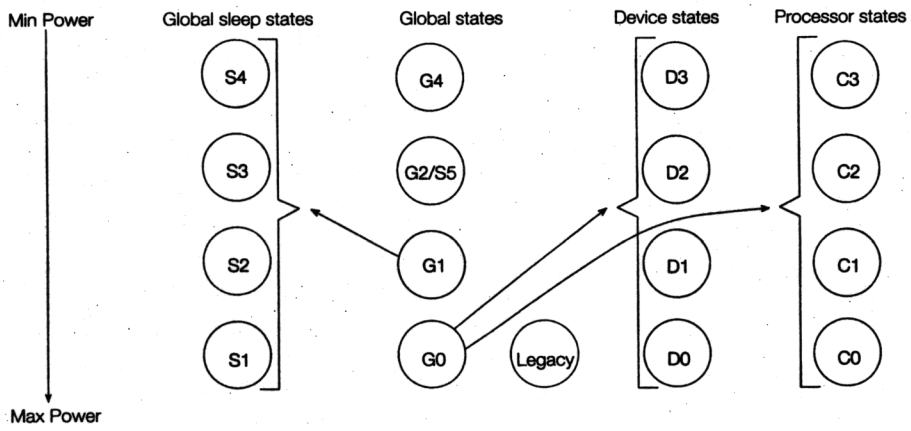

Figura 2 **Definizione deglistati operativi secondo ACPI.**

• *Stato S*1, è uno stato di *sleep* con bassa latenza di risveglio. In questo stato il sistema appare spento, ma conserva il contenuto di memorie non permanenti (RAM).
• *Stato S*2, è simile allo stato S1, con la differenza che lo stato interno della CPU (registri) e il contenuto della memoria cache sono persi. Il consumo di potenza è ridotto rispetto allo stato S1, ma un tempo più lungo è necessario per tornare allo stato attivo.
• *Stato S*3. In questo stato, tutto il contesto del sistema è perso, salvo la memoria RAM centrale. Il consumo è ulteriormente ridotto, e la latenza per ritornare allo stato attivo è ulteriormente aumentata.
• *Stato S*4, è lo stato di sleep con il minimo consumo di potenza e il più lungo tempo di risveglio. Tutte le risorse vengono spente, e un completo setup del sistema è necessario all'atto del risveglio.

La specifica ACPI definisce inoltre stati di funzionamento per tutti i componenti del sistema, suddivisi in due classi: risorse propriamente dette e CPU. Il modello a stati delle risorse definisce quattro stati di funzionamento, raffigurati in figura 2. In contrasto con gli stati globali di sistema, gli stati delle risorse non sono visibili all'utente: alcune risorse possono essere in stato di sleep, ma l'utente percepisce il sistema come completamente attivo, inoltre, diversi schemi di controllo del consumo di potenza possono essere implementati per diverse risorse.

La CPU è l'unità centrale che controlla il funzionamento del PC. ACPI definisce stati di operazione per il processore, raffigurati in figura 2. È interessante notare come vi sia una intrinseca asimmetria nel modello di sistema utilizzato da ACPI: mentre normali risorse possono essere portate in stato di *sleep* senza che l'utente percepisca un cambiamento nello stato globale del sistema, i cambiamenti di stato della CPU implicano un cambiamento macroscopico dello stato del sistema stesso, che non è più attivo. ACPI prevede anche la presenza nel sistema di unità di elaborazione secondarie, chiamate *embedded controllers*. Queste unità possono essere complessi processori, ma hanno funzioni di controllo dello stato del sistema o delle sue interfacce con il mondo esterno, e non sono disponibili per eseguire processi utente. Nonostante siano trattati come normali risorse per quanto concerne il controllo del consumo, questi componenti richiedono una interfaccia complessa per comunicare con il power manager e con altri moduli del sistema operativo.

Gli stati e le transizioni possibili in un sistema sono raffigurati nella macchina a stati gerarchica di figura 3. Durante il funzionamento normale, il sistema alterna periodi nello stato attivo (G0) e uno dei vari stati di sleep (sottostati di G1). Quando l'utente desidera spegnere il computer, si ha una transizione verso uno degli stati alla sinistra del diagramma. È importante ricordare che, mentre l'uscita da uno stato di sleep può essere quasi istantanea, la transizione di uscita da G2 e G3 richiede un completo

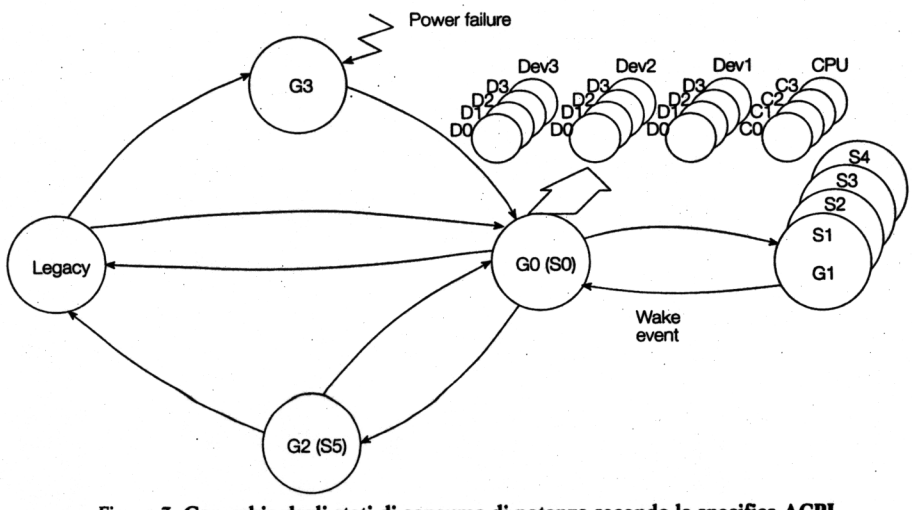

Figura 3 **Gerarchia degli stati di consumo di potenza secondo la specifica ACPI.**

te circostante (come ad esempio una chiamata telefonica, o un allarme).
• Il software e l'hardware devono cooperare per ottenere un prodotto di massima efficienza. Il programmatore del software deve avere la possibilità di osservare lo stato dell'hardware in modo semplice ed efficiente.
• Tutte le risorse hardware incluse nel sistema devono essere in grado di rispondere in modo coerente ai comandi forniti dal software. Il progettista software deve essere in grado di controllare l'hardware tramite una interfaccia standard.
Chiaramente, l'accettazione di ACPI è uno degli elementi fondamentali per il successo dell'iniziativa *OnNow*. Per questa ragione, Microsoft ha specificato che tutte le future versioni dei suoi universalmente noti sistemi operativi saranno *ACPI-compliant*.

2.2.1. ACPI In questa sezione verranno discussi i dettagli della specifica ACPI. In particolare, ci concentreremo sul modello astratto dell'hardware proposto da ACPI, di cui è importante ricordare gli obiettivi primari:
• Rendere possibile il controllo dinamico dello stato operativo e della dissipazione di potenza delle risorse hardware da parte del sistema operativo.
• Rendere i sistemi di controllo dinamico del consumo di potenza il più possibile robusti e flessibili.
• Accelerare lo sviluppo (*hardware* e *software*) di personal computers con ridotto consumo di potenza.
I principali componenti di sistema rilevanti per ACPI sono rappresentati in figura 1. Applicazioni utente interagiscono con il kernel del sistema operativo tramite normali chiamate di sistema fornite dalle *application programming interfaces* (APIs). Un modulo interno al sistema operativo (il *power manager*) implementa l'algoritmo per il controllo dinamico del consumo di potenza. Questo modulo interagisce sia con l'hardware che con il software. Possiamo definire due tipi di interazione: osservazione e controllo. L'osservazione serve per raccogliere l'informazione necessaria a prendere decisioni corrette. I meccanismi di controllo servono per influenzare direttamente lo stato delle applicazioni e delle risorse hardware. ACPI standardizza entrambi i tipi di interazione.
Il power manager usa primitive del *kernel* per osservare e

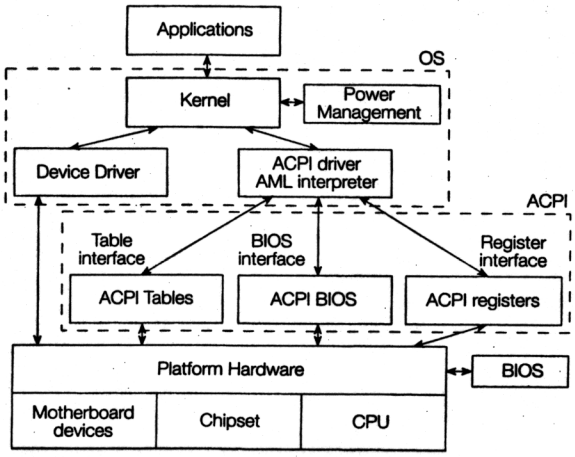

Figura 1 **Rappresentazione schematica dell'interfaccia ACPI di un personal computer.**

controllare applicazioni e risorse. Il kernel controlla l'hardware tramite moduli software noti come *device drivers*. ACPI specifica la parte di interfaccia procedurale dei device drivers necessaria per il controllo del consumo di potenza dell'hardware. Device drivers che sono in grado di interpretare comandi standard di ACPI e controllare l'hardware in modo corretto, sono detti *ACPI-compliant*. I fornitori di hardware sono tenuti a progettare e rendere disponibili *ACPI-compliant drivers* per i propri prodotti. È importante notare che il sistema operativo può interfacciarsi con componenti hardware per cui non sono disponibili *ACPI-compliant drivers*, anche se in questo caso non sarà possibile controllarne direttamente il consumo di potenza.
Nella parte inferiore di figura 1 è rappresentata in modo schematico la piattaforma hardware di un sistema ACPI-compliant. Sono visibili tre tipi di componenti. Il primo tipo corrisponde a risorse che possono essere direttamente controllate da ACPI (per esempio, controllori video, controllori di bus, hard disks, ecc.). La più importante caratteristica delle risorse è quella di poter essere individualmente controllate e poste in stato di shut-down, senza richiedere che l'intero sistema appaia inattivo. La CPU è rappresentata come una risorsa speciale, poichè la sua disponibilità è necessaria per lo svolgimento di qualsiasi attività che coinvolge il sistema operativo. L'ultimo elemento, denominato *chipset*, rappresenta tutti quei dispositivi che, come la CPU, sono generalmente allocati sulla scheda madre del PC e sono necessari al suo funzionamento. Questi dispositivi non sono controllabili direttamente come normali risorse, ma possono essere posti in stato di shut-down solo quando l'intero sistema è inattivo.
È opportuno ribadire che ACPI non specifica come implementare i componenti hardware, o quale algoritmo di controllo dinamico del consumo di potenza deve essere implementato nel power manager. Sistemi ACPI-compliant possono pertanto differenziarsi sia per l'implementazione *hardware*, sia per la politica *software* impiegata per ridurre il consumo di potenza.
Oltre a specificare i meccanismi per il controllo e l'osservazione delle condizioni del sistema, ACPI specifica un modello astratto per le risorse hardware ed il sistema stesso. Per quanto riguarda il sistema visto come una entità singola, ACPI ne specifica un modello a stati finiti, con cinque stati globali (*global states*):
• *Mechanical off* (stato G3). L'alimentazione è disconnessa e il consumo di potenza è nullo.
• *Soft off* (stato G2). Il sistema operativo deve essere ricaricato (*reboot*) per ritornare ad uno stato operativo.
• *Sleeping* (stato G1). Il sistema appare spento, ed ha un consumo di potenza ridotto. La transizione allo stato attivo richiede un tempo inversamente proporzionale alla dissipazione di potenza in stato G1.
• *Working* (stato G0). Il sistema appare accesso ed è completamente utilizzabile dall'utente.
• *Legacy*. Questo stato è utilizzato solo quando l'*hardware* non è conforme ad ACPI.
Gli stati globali del sistema sono rappresentati in figura 2, ordinati dall'alto verso il basso per consumo di potenza crescente. La specifica degli stati globali è ulteriormente raffinata dalla definizione di quattro sottostati dello stato G1, che identificano quatto diversi livelli di *sleep*:

re-boot del sistema operativo. Questa operazione può richiedere diversi minuti. Chiaramente, ACPI non specifica quando e in base a quali criteri il sistema deve cambiare di stato: queste decisioni vengono prese dalla politica di controllo implementata all'interno del power manager.

2.3. Power management e protocolli di comunicazione

Fino ad ora abbiamo focalizzato la nostra attenzione su sistemi di calcolo come i PC. Il mercato dei *personal computer* non è certamente l'unico ad essere interessato ad architetture a microprocessore con basso consumo di potenza. Al contrario, in molti casi vi sono settori in cui il controllo del consumo è fattore ancora più critico per il successo di un prodotto di quanto non lo sia per un personal computer. Per questa ragione, gli schemi dinamici per il controllo della potenza sono stati studiati in molti ambiti.

Considerevole attenzione è stata dedicata allo sviluppo di protocolli di comunicazione ad alta efficienza energetica, soprattutto per applicazioni nell'ambito della comunicazione personale (telefonia cellulare, *paging*), con lo scopo di regolare l'accesso ad un mezzo trasmissivo condiviso da multipli utenti, cercando di ottenere la massima efficienza in termini di potenza nel rispetto di una assegnata specifica di qualità del segnale.

L'esempio classico di dispositivi di comunicazione personale a basso consumo di potenza è quello del *pager*, i cui protocolli sono analizzati in [23], con particolare enfasi sull'efficienza energetica. I protocolli più efficienti si basano sul concetto di predicibilità dell'accesso, che consente al dispositivo di entrare in uno stato a bassissimo consumo per tutto il tempo in cui non sono previste richieste di comunicazione. L'introduzione di protocolli ad alta predicibilità ha consenito l'abbassamento drastico del consumo di potenza dei pagers, e di conseguenza l'incremento della durata delle batterie di questi dispositivi.

La riduzione del consumo di potenza è ovviamente un obiettivo primario anche per tutti i dispositivi di comunicazione portatile più avanzati dei pagers (telefoni cellulari, *wireless modems*, ecc.). Di conseguenza, lo studio di protocolli avanzati di comunicazione ad alta *bandwidth* e massima efficienza energetica è un'area di attiva ricerca nella quale si segnalano diversi interessanti contributi [14, 23, 29, 35]. Nel concludere la sezione, osserviamo che il campo di applicabilità delle tecniche di controllo dinamico del consumo è estremamente vasto; ci siamo qui limitati a fornire una breve panoramica delle possibili applicazioni. Nella prossima sezione tenteremo di formulare una teoria astratta che consenta di modellare molte classi di sistemi, e di descrivere e risolvere in modo generale il problema dell'ottimizzazione del consumo di potenza tramite controllo dinamico.

3. Un modello astratto

Consideriamo un generico sistema come un insieme di componenti che *generano* e/o *servono* richieste. Chiamiamo *struttura* del sistema un grafo orientato i cui nodi ed archi rappresentano rispettivamente i componenti e le possibili interazioni (scambi di richieste) tra essi. Con riferimento ad un arco del grafo, chiamiamo *service requester* (SR) il componente associato al nodo da cui l'arco esce e *service provider* (SP) il componente associato al nodo in cui l'arco entra. L'arco rappresenta la possibilità che il SR richieda servizi al SP.

I grafi di figura 4 descrivono la struttura di due semplici sistemi: il primo (Figura 4A) rappresenta l'interazione tra un singolo SR e un singolo SP, il secondo (Figura 4B) rappresenta le interazioni tra alcuni dei componenti di un semplice personal computer. Mouse e tastiera generano richieste per la CPU, che a sua volta genera richieste per display, interfaccia di rete e hard disk. L'interfaccia di rete può a sua volta generare richieste per la CPU. Si noti che i due grafi di figura 4 possono essere visti come due rappresentazioni dello stesso sistema a diversi livelli di dettaglio. Supponiamo, ad esempio, che l'hard disk sia il solo componente *power manageable* di cui ci interessa conoscere le statistiche d'uso. In questo caso lo schema di figura 4B può essere ridotto a quello di figura 4A associando l'hard disk al SP e tutto il resto del sistema al SR, poichè l'unica funzione del sistema che ci interssa modellare ai fini del *power management* è la generazione di richieste per il disco.

Dal punto di vista del *power management* i componenti del sistema sono visti come *macchine a stati finiti*. Ogni componente può trovarsi in un numero finito di stati di funzionamento caratterizzati, per un SR, dal numero medio di richieste generate nell'unità di tempo o, per un SP, dal tempo medio di servizio e dal consumo di potenza. I SR sono sorgenti di traffico. Se un SR rappresenta un dispositivo di *input* per il sistema in esame, il suo comportamento in ogni stato e le transizioni tra gli stati dipendono dall'ambiente esterno. Le transizioni tra gli stati di un SP *power manageable* sono invece controllate dal *power manager*. Il sistema, nel suo complesso, è una macchina a stati finiti il cui stato è il prodotto cartesiano degli stati dei singoli componenti.

L'alto livello di astrazione rende difficile, se non impossibile, avere informazioni precise sul traffico, sul consumo di potenza e sulle prestazioni istantanee di ogni risorsa. L'incertezza che ne consegue rende appropriato modellare tali grandezze come variabili aleatorie di cui considerare i valori medi e le distribuzioni. In generale, il modello di un componente è una macchina a stati finiti non deterministica in cui le grandezze associate ad ogni stato sono valori medi di variabili aleatorie e le transizioni tra gli stati sono non deterministiche.

I modelli stocastici si prestano a modellare sia la mancanza di informazioni dettagliate dovute all'astrazione della funzionalità, sia le fluttuazioni delle grandezze osservate dovute a fattori ambientali di cui non è possibile tener

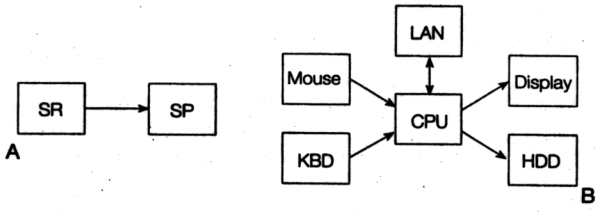

Figura 4 **Esempi di rappresentazioni strutturali di sistemi elettronici. A Generica interazione tra un service requester e un service provider. B Interazione tra le risorse di un personal computer.**

conto a priori. La ricerca di strategie ottime per il controllo dinamico del consumo di potenza diventa un problema di *controllo stocastico ottimo* [25], la cui soluzione e reale utilità dipendono fortemente dalle ipotesi semplificative introdotte nel modello durante il processo di astrazione.

3.1. Il modello stocastico: un esempio

In questa sottosezione analizziamo in dettaglio un semplice esempio di applicazione del modello stocastico, rappresentato in figura 5, composto da un SP che riceve richieste da un SR attraverso una coda (Q) ed è controllato da un *power manager* (PM). Tale modello è adeguato, ad esempio, a rappresentare e studiare sistemi quali utente-computer, o *CPU-hard disk.*

Ci riconduciamo all'ipotesi di *tempo discreto* osservando il sistema solo ad intervalli di tempo regolari e assumiamo che ogni cambiamento di stato sia sincronizzato con gli istanti di discretizzazione (in pratica, in ogni intervallo di tempo ogni componente può trovarsi in un solo stato). Modelliamo i componenti come catene di *Markov parametriche* [27, 34], assumendo che la storia passata del sistema sia rappresentata completamente dal suo stato presente e che solo questo ne influenzi l'evoluzione futura. Le probabilità di transizione dipendono esclusivamente dallo stato presente e, nel caso di componenti *power manageable*, da un parametro che rappresenta il comando inviato dal PM. I modelli dei singoli componenti sono descritti di seguito.

Service provider (*SP*). È un dispositivo (ad esempio un *hard disk driver*) che riceve e serve richieste provenienti da un SR. Il modello è una catena di Markov i cui stati sp (appartenenti all'insieme $\{1,2,..., S_p\}$) rappresentano i diversi stati di funzionamento caratterizzati da valori noti di prestazioni e consumo di potenza. Chiamiamo *attivi* gli stati in cui la prestazione è non nulla, *inattivi* gli altri. Nel caso più semplice il modello ha due stati ($S_p = 2$), uno attivo e uno inattivo, che possiamo chiamare *on* e *off*. In generale gli stati possono essere più di due, come nel caso di componenti conformi alla specifica ACPI, mentre modelli ad un solo stato sono privi di interesse ai fini del *power management*. Le transizioni tra gli stati sono controllate dal PM attraverso un comando a appartenenete all'insieme $A = \{1, 2,..., N_a\}$. Nel caso di un *provider* a due stati i comandi possono essere ad esempio "accendi" (*s_on*) e "spegni" (*s_off*). La risposta del SP ai comandi è non-deterministica: benchè dal comando dipendono le probabilità di transizione dallo stato presente allo stato futuro, in generale da ogni stato esistono diverse possibili evoluzioni con probabilità non nulla. Ad esempio, a fronte del commando *s_on* la probabilità di transizione da *off* a *on* potrebbe essere 0.8, lasciando al SP una probabilità non nulla di restare *off* nel successivo intervallo di tempo. Questa forma di non determinismo permette di modellare l'incertezza nei tempi di transizione tra stati causata dall'astrazione delle informazioni funzionali. Come vedremo in seguito, il comando può influenzare anche prestazioni e consumo, che diventano pertanto funzioni di s_p ed a.

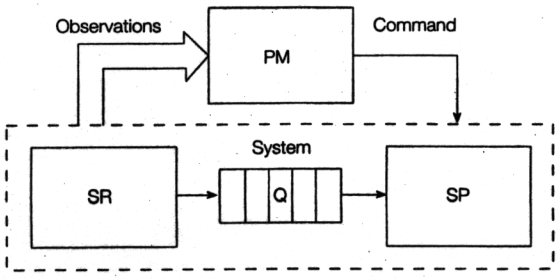

Figura 5 **Struttura di riferimento per la costruzione di un modello operativo.**

Service requester (*SR*). È una sorgente autonoma di richieste. Il modello è una catena di Markov non parametrica i cui stati s_r (appartenenti all'insieme $\{0,1,...,S_r - 1\}$) rappresentano il numero di richieste generate dal SR in un intervallo di tempo.

Coda (*Q*). Memorizza temporaneamente le richieste del SR in attesa che il SP sia disponibile a servirle. Dal punto di vista strutturale la coda ha la doppia funzione di SP e di SR, poichè genera e riceve richieste. Se L è la massima lunghezza della coda, il modello astratto è una catena di Markov a $S_q = L + 1$ stati $s_q = \{0,1,...,S_q - 1\}$ associati alle possibili condizioni di riempimento. Le transizioni tra gli stati dipendono dallo stato del SR (che determina la frequenza di riempimento della coda), dallo stato del SP (che determina la velocità di smaltimento delle richieste in coda) e dai comandi del PM.

Power Manager (*PM*). Osserva lo stato del sistema e invia comandi al SP per controllarne dinamicamente lo stato di funzionamento secondo una prefissata *strategia*. Assumiamo che il consumo di potenza del PM sia trascurabile rispetto a quello del SP.

Il sistema formato da SP, SR e Q e controllato dal PM è il prodotto di tre catene di Markov. Pertanto è a sua volta una catena di Markov con spazio degli stati di dimensione $S = S_r\, S_p\, S_q$, il cui stato è una terna $s = (s_r, s_p, s_q)$ e la cui matrice di transizione dipende dal comando a del PM.

A titolo di esempio consideriamo il sistema di figura 6 il cui SP è un HDD *power manageable*. Il SR ha solo due stati, 0 e 1, che rappresentano il numero di richieste generate in ogni intervallo di tempo. La coda ha lunghezza 1 ed è rappresentata da una catena a due stati, 0 e 1, che rappresentano il numero di richieste da servire. Anche il SP ha due stati, *on* e *off*: nello stato *on* serve al più una richiesta in ogni periodo, con un consumo di potenza di 3 W; nello stato *off* non serve alcuna richiesta e non consuma potenza. Alle transizioni tra i due stati è associato un consumo di 4 W (superiore al consumo dello stato attivo). Vale la pena di precisare cosa rappresenta (e come viene modellato) il consumo di potenza associato alle transizioni. Infatti, in accordo con il modello di Markov tempo-discreto, le transizioni tra gli stati sono istantanee ed è quindi irrilevante il consumo di potenza ad esse associato. Nella nostra astrazione, tuttavia, le transizioni non istantanee sono rappresentate da probabilità di transizione non unitarie. Si consideri ad esempio la transizione tra lo stato *off* e lo stato *on* del SP di figura 6 quando il comando è *s_on*. La probabilità di compiere istantaneamente la transizione è 0.1, mentre con probabilità 0.9 il SP resta nello stato inattivo. Questo significa che in media è necessario ribadire il comando per 10 volte prima che lo stato attivo venga effettivamente raggiunto. Dal punto di vista delle prestazioni, questo permette di modellare un tempo di transizione 10 volte superiore al perio-

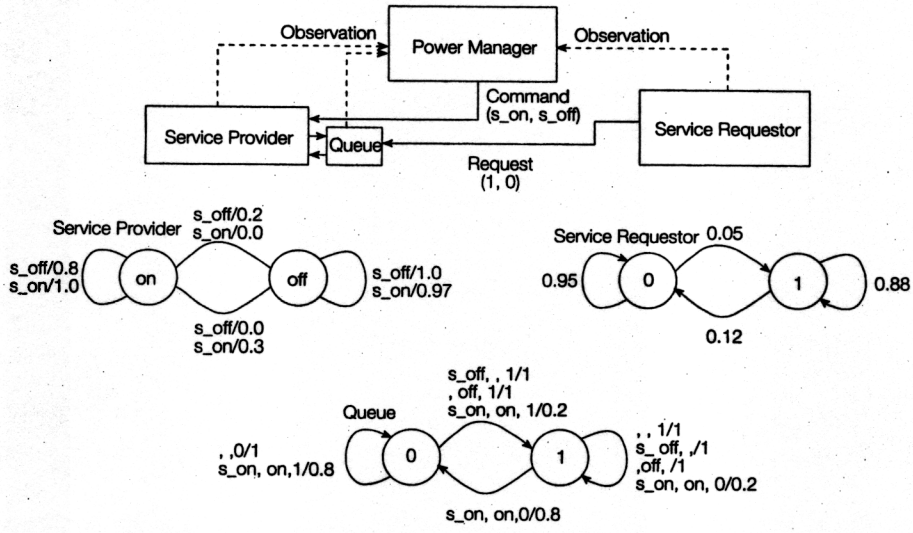

Figura 6 **Esempio di modello stocastico di un sistema power manageable.**

do di discretizzazione. Per tener conto del consumo di potenza associato a questa transizione non istantanea modelliamo il consumo di potenza del SP come una funzione che dipende sia dallo stato del SP che dal comando. Il consumo è nullo quando lo stato è *off* e il comando è *s_off*, ma il consumo è 4 W quando lo stato è *off* e il comando è *s_on*.
Il SR evolve autonomamente, mentre lo stato futuro del SP dipende dal comando e quello della coda sia dal comando che dagli stati di SP ed SR. Consideriamo, ad esempio, il SP nello stato attivo. Quando il comando è *s_on* il disco resta acceso con probabilità 1, mentre quando il comando è *s_off* la probabilità di spegnimento è 0.2 (cioè avviene una transizione verso lo stato *off* che in media richiede cinque intervalli di tempo).

4. Ottimizzazione di politiche di power management

In questa sezione facciamo riferimento al modello astratto introdotto nella sezione precedente per affrontare il problema di ottimizzazione delle politiche di controllo dinamico del consumo di potenza. La politica ottima è l'incognita di un problema di ottimizzazione condizionata che ammette due istanze duali: minimizzazione del consumo per un dato requisito di prestazioni, massimizzazione delle prestazioni per un dato requisito di consumo. Senza perdita di generalità, nel seguito descriviamo il procedimento di soluzione della prima istanza del problema, rimandando a [26] per maggiori dettagli.
Analizziamo innanzitutto il modo in cui il PM controlla il sistema per definire formalmente il concetto di politica e individuare lo spazio delle possibili soluzioni. In ogni periodo di tempo, il PM osserva la storia del sistema e controlla il SP prendendo una *decisione* e inviando un *comando*. La decisione può essere *deterministica o randomizzata*. Nel primo caso il comando da inviare è una funzione deterministica della storia passata, nel secondo viene scelto in modo casuale in base a distribuzioni di probabilità dipendenti dallo stato del sistema. Le decisioni deterministiche sono casi particolari di decisioni randomizzate, in cui un comando ha probabilità 1 e gli altri 0.
Una *politica* è una sequenza di decisioni, ed è *stazionaria* se in ogni intervallo di tempo ad osservazioni uguali corrispondono decisioni uguali. In pratica, la stazionarietà indica che la relazione tra osservazione e decisione non cambia nel tempo, ma non esclude che decisioni diverse possano essere prese a fronte di osservazioni diverse. Una politica stazionaria è detta *markoviana* se la decisione dipende solo dallo stato presente del sistema.
È stato dimostrato formalmente che il problema di ottimizzazione delle politiche di controllo dinamico del consumo di potenza di un sistema per il quale valga il modello introdotto in sezione 3 ammette soluzione unica, e che questa è una politica stazionaria markoviana [25, 26]. Vale la pena di sottolineare che le politiche markoviane sono particolarmente semplici da implementare, in quanto richiedono soltanto l'osservazione istantanea dello stato presente del sistema. In generale la politica ottima è randomizzata, ma è deterministica nel caso particolare di ottimizzazione non condizionata o di ottimizzazione con condizioni inattive [25].
La politica ottima può essere individuata in tempo polinomiale con tecniche di programmazione lineare (LP). La descrizione intuitiva della formulazione del problema si basa su una semplice osservazione: l'effetto di una politica sul sistema può essere espresso nei termini delle *frequenze* di occorrenza delle possibili coppie (*stato, comando*). Tali frequenze indicano la probabilità che in un dato intervallo di tempo il sistema si trovi in un certo stato e il PM invii un certo comando. Il modello di Markov del sistema determina le *leggi di conservazione* che le frequenze di stato e comando devono soddisfare. In particolare, la somma di tutte le frequenze deve essere uno e la probabilità che lo stato corrente sia x deve essere uguale alla probabilità che x sia lo stato iniziale del sistema più la probabilità di raggiungere x da un qualsiasi altro stato. I valori medi delle funzioni di costo e prestazione (che rappresentano rispettivamente l'obiettivo e la condizione del problema di ottimizzazione) sono espressioni lineari delle frequenze.
Le frequenze di stato e comando dipendono dalla politica. Noti la politica e il modello del sistema, le frequenze sono univocamente determinate e di conseguenza lo sono i va-

lori attesi delle funzioni di costo e prestazioni. Se invece la politica è incognita, anche le frequenze lo sono. Quindi, il problema di ottimizzazione delle politiche di *power management* può essere espresso come il problema di minimizzazione di una funzione lineare delle frequenze di (*stato, azione*) condizionato a vincoli lineari. La soluzione (unica) del problema è il valore di tutte le frequenze, dalle quali è possibile ricavare la politica ottima.

I programmi lineari associati al problema di ottimizzazione di politiche di *power management* possono essere risolti per sistemi di interesse pratico utilizzando *software* standard. L'ottimizzatore descritto in [26] affida la soluzione del problema LP a PCx, un solutore di problemi LP basato su un algoritmo di *interior point* [15].

Nella figura 7 è riportata la curva di Pareto sul piano prestazioni-consumo ottenuta risolvendo ripetutamente (con diversi vincoli sulle prestazioni) il problema di ottimizzazione per il sistema di figura 6. Le prestazioni sono espresse in termini di lunghezza media della coda, che è il tempo medio di attesa per una richiesta. Una condizione aggiuntiva, detta *request loss*, è stata utilizzata per rappresentare la massima probabilità di perdita di una richiesta a causa della condizione di coda piena. È interessante osservare come il compromesso tra prestazioni e consumo di potenza sia influenzato dalla condizione addizionale. In particolare, se deve essere garantito un *request loss* minore di 0.1337, il SP non può mai essere spento ed è impossibile risparmiare potenza indipendentemente dal requisito di prestazione minima.

Nella figura 8 è riportata la curva di Pareto per un sistema più complesso. Il SP è un HDD *power manageable* disponibile in commercio, con uno stato attivo e quattro inattivi che offrono diversi compromessi tra consumo e tempo di transizione da e verso lo stato attivo [41]. Il consumo medio del disco nello stato attivo è 2.5 W. Il modello del SR è una catena di Markov a due stati caratterizzata su una traccia di accessi a disco disponibile in rete [40]. Per l'esperimento è stata utilizzata una coda di lunghezza 2. Sul piano prestazioni-consumo sono riportati anche i punti rappresentativi di diverse politiche euristiche. Alcune di esse realizzano compromessi non distanti dalla curva di Pareto e non è escluso che progettisti esperti possano con-

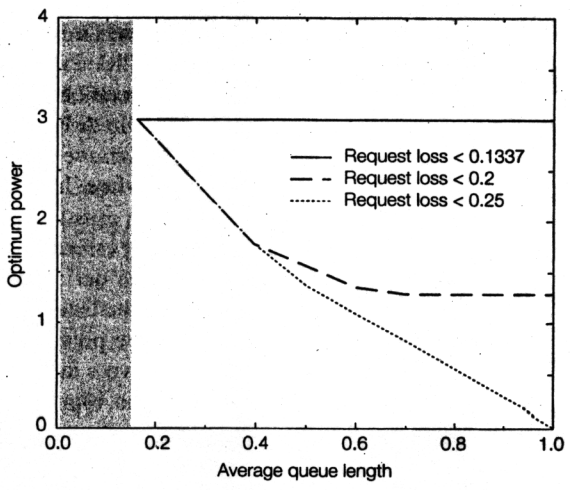

Figura 7 **Possibili punti di compromesso tra prestazioni e consumo per il sistema di figura 6.**

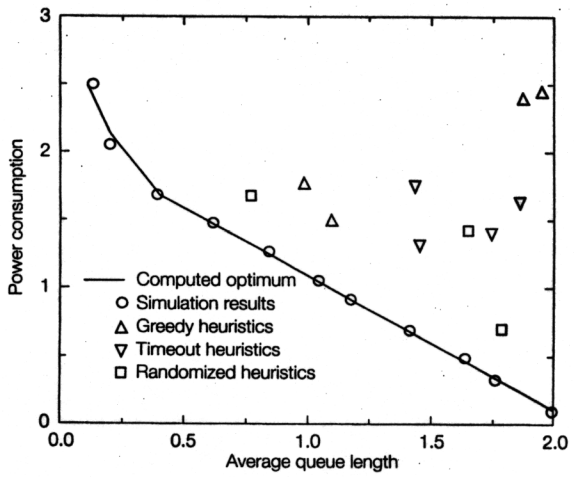

Figura 8 **Possibili compromessi tra prestazioni e consumo per un hard disk power manageable disponibile in commercio.**

cepire politiche ottime. Tuttavia, le politiche euristiche presentano due inconvenienti principali rispetto a quelle prodotte come soluzione esatta di problemi LP. Primo, non forniscono criteri generali per soddisfare eventuali requisiti di prestazione, nè è sempre possibile adottare approcci iterativi basati su tentativi e verifiche a causa dell'elevato numero di parametri (nel nostro caso la politica è rappreserntata da una matrice di dimensione 66 × 5 con 330 elementi). Secondo, benchè sia possibile produrre politiche euristiche che forniscono risultati accettabili, non c'è modo di sapere se tali risultati sono migliorabili.

Pur avendo illustrato come il modello markoviano consenta una formulazione rigorosa e una soluzione esatta del problema di ottimizzazione delle politiche, dobbiamo ricordare che non è sempre possibile assumerne la validità in sistemi reali. Vi sono casi in cui la distribuzione del carico di lavoro o dei tempi di servizio non può essere modellata come una catena di Markov, nemmeno in prima approssimazione. In queste condizioni, le politiche markoviane debbono essere viste come soluzioni euristiche, che possono essere superate da altre politiche.

Tra le politiche euristiche, l'approccio senz'altro più diffuso è quello basato su *timeout*, che può essere riassunto come segue. Non appena il carico del sistema (o del componente) diventa nullo (o si riduce al di sotto di una soglia minima), viene attivato un *timer*. Quando il valore del *timer* supera una soglia predeterminata T_{TO} (il *timeout*), senza che il carico sia aumentato, il sistema viene portato in stato di *sleep* per risparmiare potenza. È interessante notare come questa politica si basi sull'assunzione che se il tempo di inattività è più lungo del *timeout*, allora si può assumere che tale tempo sarà lungo abbastanza per ammortizzare (grazie al risparmio di potenza) il costo della transizione allo stato di *sleep*. Tale assunzione è in contrasto con quella di carico markoviano, secondo la quale la probabilità di restare più a lungo inattivi dato che si è rimasti inattivi per un intervallo di tempo T_{TO} è rigorosamente costante per qualsasi valore di T_{TO}.

È possibile ottimizzare una politica di *timeout* iterando su molti diversi valori T_{TO} e scegliendo quello che porta al compromesso più accettabile tra prestazioni e potenza. Questa strategia ha però applicabilità limitata nel caso di

sistemi con multipli stati di *sleep*, o di sistemi con code o *buffers*. In questi casi più complessi, la scelta di valori ottimi di *timeout* è tutt'ora un problema aperto.
Un limite intrinseco delle politiche di *timeout* è il fatto che forzino il sistema ad attendere in stato attivo per T_{TO}, sprecando potenza. Per ovviare a questa limitazione, sono state sviluppate diverse classi di politiche *predittive* [22, 30]. L'assunzione di base in questi approcci è che sia possibile costruire un "oracolo" in grado di predire la durata di un periodo di inattività. In caso di predizione perfetta, sarebbe sufficiente consultare l'oracolo all'inizio di ogni periodo di inattività, e prendere una decisione deterministica basata sulla predizione. A differenza dell'approccio basato su *timeout*, nessuna potenza sarebbe sprecata, visto che la decisione viene presa proprio all'inizio del periodo di inattività. Ovviamente è impossibile, salvo in casi molto particolari come quello di carico periodico, predire esattamente la durata dei periodi di inattività. Le politiche predittive si differenziano per gli algoritmi di predizione adottati. In generale, le predizioni sono basate sull'osservazione della storia passata. Il lettore interessato allo studio di politiche predittive è invitato a fare riferimento a [22, 30] e alla dettagliata analisi e bibliografia presentata in [20].

4.1. Implementazione

Il PM che implementa la politica di controllo del consumo di potenza può essere sia un componente *hardware* sia una procedura *software*. In alcuni casi può essere conveniente implementare il PM come un circuito di controllo i cui ingressi sono i segnali che rappresentano lo stato del sistema, mentre le uscite sono i segnali di controllo che rappresentano i comandi. Se la politica è espressa nei termini di un linguaggio di descrizione dell'*hardware* (HDL), l'implementazione può essere generata automaticamente avvalendosi dei tradizionali strumenti di sintesi.
Una politica deterministica è un semplice schema di *look-up* che può essere implementato con un circuito logico combinatorio o con una memoria non volatile che memorizzi i comandi corrispondenti ad ogni stato del sistema. L'implementazione di politche basate su *timeout* presuppone l'esistenza di dispositivi in grado di generare i *timeout* richiesti. Tali dispositivi, noti come *real-time clocks* o *timers*, sono disponibili in commercio a basso costo e minima dissipazione di potenza.
Le politiche randomizzate richiedono invece la memorizzazione di una tabella bidimensionale, che ad ogni stato del sistema associa le probabilità condizionate di ogni comando, e la generazione di un numero pseudocasuale per la scelta del comando, generato ad esempio da un registro a scorrimento a retroazione lineare (LFSR). In questo caso, se le probabilità condizionate sono normalizzate alla lunghezza del registro, la scelta del comando è effettuata mediante un semplice confronto. Se ci sono solo due possibili comandi (ad esempio *s_on* e *s_off*) la somma delle loro probabilità condizionate è 1 e solo una delle due probabilità deve essere memorizzata. Il risultato binario del confronto tra tale probabilità e il valore corrente del registro a scorrimento rappresenta direttamente il comando da impartire. Tale schema può essere semplicemente esteso al caso di N_a comandi utilizzando una tabella con $N_a - 1$ elementi per stato ed effettuando (eventualmente in parallelo) $N_a - 1$ confronti con il numero pseudocasuale.
L'implementazione *software* del PM richiede la codifica della politica nei termini di un linguaggio di programmazione ad alto livello, la sua compilazione e il suo inserimento nel sistemo operativo. Nel caso di politiche randomizzate la routine di *power management* può utilizzare un generatore *software* di numeri pseudocasuali per decidere il comando da impartire. La routine di PM può essere eseguita in *kernel mode* e sincronizzata all'esecuzione del *task scheduler* del sistema operativo per non aumentare il numero di cambiamenti di contesto.

4.2. Politiche adattative

Fin qui abbiamo implicitamente assunto che la politica di *power management* potesse essere ottimizzata una volta per tutte ed implementata staticamente in *hardware* o in *software*. Sfortunatamente, l'ottimalità di una politica dipende dalla statistica di uso del sistema, rappresentata nella nostra astrazione dal modello dal SR. Se le statistiche d'uso non sono note a priori o sono non stazionarie, non è possibile individuare una politica stazionaria ottima che realizzi il miglior compromesso tra prestazioni e consumo a fronte di ogni possibile situazione operativa. In tal caso è necessario ricorrere a politiche adattative. Ci sono diversi modi per concepire e implementare politiche non stazionarie, tra cui la soluzione più elementare consiste nell'utilizzare diverse politiche statiche per le diverse condizioni operative. Benchè le singole politiche possano essere progettate ed implementate come descritto nelle sezioni precedenti, la non stazionarietà pone due nuovi problemi: la rilevazione dei cambiamenti statistici delle condizioni di carico e il cambiamento dinamico della politica. In accordo con il modello astratto proposto in sezione 3, la statistica dell'utente può essere rappresentata dalle probabilità di transizione della catena di Markov del SR.
Nel caso di una catena a due stati, i parametri statistici sono solo due (la probabilità di avere richieste in un dato intervallo di tempo condizionata al fatto che ce ne fossero o no nell'intervallo precedente) e possono essere stimati *on-line* osservando il flusso di richieste. Brusche variazioni di tali parametri possono essere interpretate come indici di non stazionarietà dell'utente ed utilizzate per cambiare politica di *power management*, calcolandone una nuova *on-line*, o scegliendone una da una libreria di politiche precedentemente ottimizzate *off-line*. Nel primo caso occorre risolvere un nuovo problema di ottimizzazione condizionata, il che richiede l'integrazione dell'ottimizzatore nel sistema e può comportare un sensibile calo di prestazioni. Nel secondo caso diverse politiche possono essere ottimizzate una volta per tutte per diverse condizioni di carico e poste in una memoria indirizzata dai parametri statistici che rappresentano le condizioni di carico attuali. Nel caso di politiche predittive, l'adattamento avviene cambiando dinamicamente la predizione della durata dei periodi di inattività in base alla storia passata. Una ampia varietà di schemi di adattamento è descritta nei lavori di Golding [19, 20]. La difficoltà principale nell'applicazione di questi schemi sta nella scelta

della "lunghezza" della memoria del predittore. In termini intuitivi, si tratta di scegliere quanto a lungo ed in che termini eventi passati possono influenzare la predizione di eventi futuri.

4. Conclusioni

Sistemi a microprocessore complessi sono sovente sottoutilizzati o parzialmente inattivi. È possibile aumentare l'efficienza energetica di tali sistemi controllandone dinamicamente lo stato di funzionamento. Benchè tecniche di controllo dinamico del consumo di potenza siano già largamente applicate a sistemi di diversa natura, esse si basano principalmente su politiche euristiche e su implementazioni *ad-hoc*. La definizione di standard industriali quali *OnNow* e ACPI favorirà nell'immediato futuro lo sviluppo di sistemi a microprocessore in cui il controllo dinamico del consumo di potenza è affidato al sistema operativo. Questo renderà generale e flessibile l'implementazione delle strategie di *power management*, ma non risolverà il problema della loro ottimizzazione.
In questo lavoro abbiamo introdotto un modello astratto per sistemi *power manageable* che permette di formulare e risolvere rigorosamente il problema di ottimizzazione condizionata delle politiche di *power management* con tecniche di programmazione lineare. Con la crescente diffusione di sistemi elettronici portatili e la nuova attenzione al consumo di potenza di sistemi anche non portatili, i progettisti si troveranno sempre più spesso ad affrontare il problema della ricerca di soluzioni di controllo dinamico del consumo. Le tecniche di stima e ottimizzazione automatica presentate in questo lavoro rappresentano un primo passo verso lo sviluppo di strumenti CAD per il progetto di sistemi a controllo dinamico.

BIBLIOGRAFIA

[1] Nadel B: The Green Machine. *PC Magazine*, Vol 12, n. 10, May 25, 1993, p.110.

[2] Udani S, Smith J: *The Power Broker: Intelligent Power Management for Mobile Computing*. Technical report MS-CIS-96-12, Dept. of Computer Information Science, University of Pennsylvania, May 1996.

[3] Chandrakasan A, Brodersen R: *Low-Power Digital CMOS Design*. Kluwer, 1995.

[4] Monteiro J, Devadas S: *Computer-Aided Techniques for Low-Power Sequential Logic Circuits*. Kluwer, 1997.

[5] Rabaey JM, Pedram M (editors): *Low-Power Design Methodologies*. Kluwer, 1996.

[6] Sakurai T, Kuroda T: Low-Power Circuit Design for Multimedia CMOS VLSI. In *Workshop on Synthesis and System Integration of Mixed Technologies*, Nov. 1996, p. 3-10.

[7] Benini L, De Micheli G: *Dynamic Power Management of Circuits and Systems: Design Techniques and CAD Tools*. Kluwer, 1997.

[8] Lorch J, Smith A: Software Strategies for Portable Computer Energy Management. *IEEE Personal Communications Magazine*, June 1998.

[9] Kapadia H, De Micheli G, Benini L: *Reducing Switching Activity on Datapath Buses with Control-Signal Gating*. In Proceedings of the Custom Integrated Circuit Conference, 1998, p. 589-592.

[10] Benini L, De Micheli G: Automatic Synthesis of Low-Power Gated-Clock Finite-State Machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, n. 6, June 1996, p. 630-643.

[11] Benini L, Siegel P, De Micheli G: Automatic Synthesis of Gated Clocks for Power Reduction in Sequential Circuits. *IEEE Design and Test of Computers*, Dec. 1994, p. 32-40.

[12] Benini L, Bogliolo A, Cavallucci S, Riccò B: *Monitoring System Activity for OS-directed Dynamic Power Management*. In Proceedings International Symposium on Low-Power Design, 1998.

[13] Benini L, Hodgson R, Siegel P: *System-Level Power Estimation and Optimization*. In Proceedings of Int.l Symposium of Low-Power Electronics and Design, 1998.

[14] Rulnick J, Bambos N: Mobile Power Management for Wireless Communication Networks. *Wireless Networks*, Vol. 3, n. 1, 1997, p. 3-14.

[15] Czyzyk J, Mehrotra S, Wright S: *PCx User Guide*. Technical Report OTC 96/01, Optimization Technology Center, May, 1996.

[16] Debnath G, Debnath K, Fernando R: *The Pentium Processor-90/100, Microarchitecture and Low-Power Circuit Design*. In International conference on VLSI design, Jan. 1995, p. 185-190.

[17] Furber S: *ARM System Architecture*. Addison-Wesley, 1997.

[18] Gary S, Ippolito P, et al.: PowerPC 603, a Microprocessor for Portable Computers. *IEEE Design \& Test of Computers*, Vol. 11, n. 4, Win. 1994, p. 14-23.

[19] Golding R, Bosch P, Wilkes J: *Idleness is not Sloth*. In Proceedings of Winter USENIX Technical Conference, 1995, p. 201-212.

[20] Golding R, Bosh P, Wilkes J: *Idleness is not Sloth*. HP Laboratories Technical Report HPL-96-140, 1996.

[21] Gowan M, Biro L, Jackson D: *Power Considerations in the Design of the Alpha 21264 Microprocessor*. In Proceedings of the Design Automation Conference, 1998, p. 726-731.

[22] Hwang C-H, Wu A: *A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation*. In Proceedings of the Int.l Conference on Computer Aided Design, 1997, p. 28-32.

[23] Mangione-Smith B: Low-Power Communication Protocols: Paging and Beyond. *IEEE Symposium on Low-Power Electronics*, 1995, p. 8-11.

[24] Nebel W, Mermet J, (Eds.): *Low-Power Design in Deep Submicron Electronics*. Kluwer. 1997.

[25] Puterman M: *Finite Markov decision processes*. Wiley, 1994.

[26] Paleologo G, Benini L, Bogliolo A, De Micheli G: *Policy Optimization for Dynamic Power Management*. In Proceedings of the Design Automation Conference, 1998, p. 182-187.

[27] Papoulis A: *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 1984.

[28] Rosenblum M, Bugnion E, Devine S, Herrod SA: Using the SimOS Machine Simulator to Study Complex Computer Systems. *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, n. 1, 1997, p. 78-103.

[29] Sivalingham K, Srivastava M, et al.: *Low-power Access Protocols Based on Scheduling for Wireless and Mobile ATM Networks*. Int.l Conference on Universal Personal Communications, 1997, p. 429-433.

[30] Srivastava M, Chandrakasan A, Brodersen R: Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation. *IEEE Transactions on VLSI Systems*, Vol. 4, n. 1, March 1996, p. 42-55.

[31] Tiwari V, Singh D, Rajgopal S, Metha G, Patel R, Baez F: *Reducing Power in High-Performance Microprocessors*. In Proceedings of the Design Automation Conference, 1998, p. 732-737.

[32] Nielsen LS, Niessen C, SparsVo J, Van Berkel K: Low-Power Operation Using Self-Timed Circuits and Adaptive Scaling of the Supply Voltage. *IEEE Transactions on VLSI Systems*, Vol. 2, n. 4, Dec. 1994, p. 391-397.

[33] Torvalds L: *Linux Kernel Implementation*. In Proceedings of Open Systems. Looking into the future, 1994, p. 9-14.

[34] Trivedi K: *Probability and Statistics with Reliability, Queueing and Computer Science Applications*. Prentice Hall, 1982.

[35] Zorzi M, Rao R: Energy-Constrained Error Control for Wireless Channels. *IEEE Personal Communications*, Vol. 4, n. 6, Dec. 1997, p. 27-33.

[36] http://www.intel.com/ial/powermgm/specs.html, Intel, Microsoft and Toshiba. *Advanced Configuration and Power Interface specification*, Dec. 1996.

[37] http://developer.intel.com/IAL/powermgm/apmovr.htm, Intel. *Advanced Power Management Overview*, 1998.

[38] http://www.microsoft.com/hwdev/pcfuture/ONNOW.HTM, Microsoft. *OnNow: the evolution of the PC platform*, Aug. 1997.

[39] http://www.storage.ibm.com/storage/oem/data/travvp.htm. *Technical specification of hard-drive IBM Travelstar VP 2.5-inch*, 1996.

[40] http://now.cs.berkeley.edu/Xfs/AuspexTraces/auspex.html: *Auspex File System Traces*. 1993.

[41] http://www.storage.ibm.com/storage/oem/data/travvp.htm. *Hard Drive IBM Travelstar VP 2.5-inch*, 1996.