Latency Tolerance

Introduction

The processor-memory gap and the processor-I/O gap increases by more than a factor of five per decade. Thus, the latency of memory accesses and I/O-accesses are increasing over time.

In distributed memory systems, the latency of the network, network interface, and the end-point processing is added to that of accessing the local memory of the node.

Latency usually grows with the size of the machine since more nodes implies more communication relative to computation, more hops in the network for general communication, and likely more contention.

One goal of the communication protocols, the programming model, and the application algorithms is to reduce the frequency of long-latency events. A major part of parallel programming consists of structuring the application in a way to reduce the frequency of high-latency accesses.

The remaining events can be handled in two ways:

- **latency minimization**
- **latency hiding**

Parallel Architecture can be classified by these to methods (s.a. Introduction).

Latency minimization is the first goal to achieve. It can be done by optimizing the memory hierarchy for load and store, optimize the communication operations performed by the NIC, and provide support for synchronization.

Nevertheless, there is a remaining latency which cannot be minimized any further without massively increase hardware costs.

The approach is to tolerate the remaining latency; that is, hide the latency from the processor's critical path by overlapping it with computation or other high latency operations. It is important to recognize that this latency hiding requires independent work as mentioned in our analysis of parallelism.

The Latency of a memory access or communication operation includes all components of the time that elapses from issue by the processor until completion. For communication, this includes processor overhead, assist occupancy (NIC), transit delay, bandwidth-related costs, and contention. The latency may be for a one-way transfer or a two-way (roung-trip) transfer.

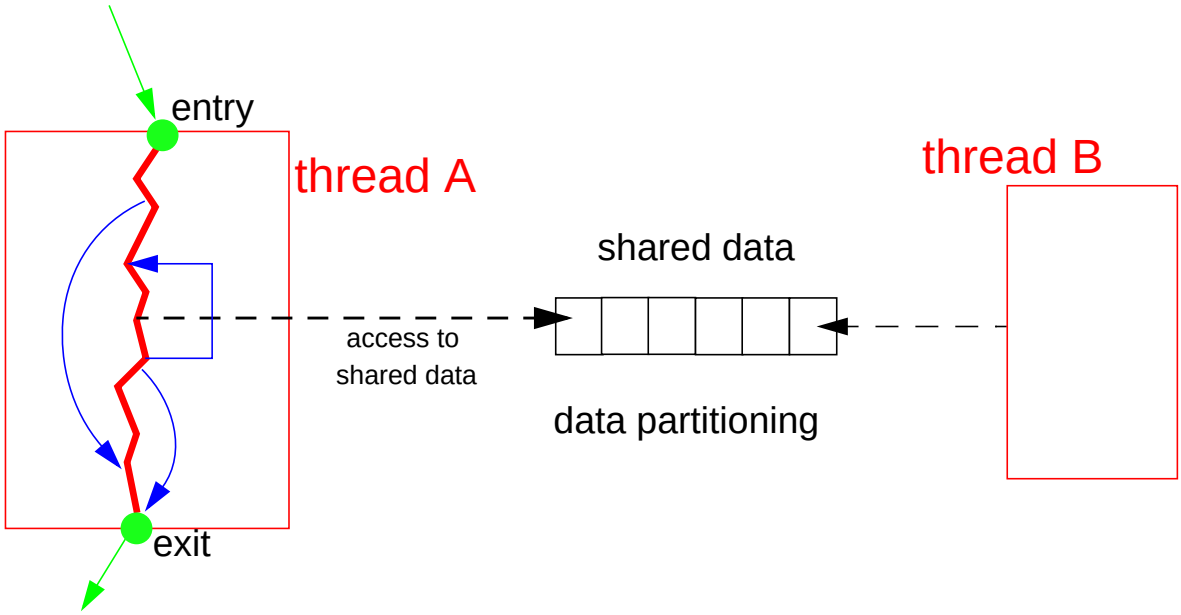
Multithread Architectures (MTA)

Mehrfädige Architekturen

Das Operationsprinzip der mehrfädigen Architekturen (‘multithread architektures’ MTA) basiert auf der Kontrollstruktur des sequentiellen Kontrollablaufs und der Informationsstruktur von konventionellen von Neumann Variablen und zum Teil auf Variablen mit zukünftigem Wert (‘futures’) [futures]. Sie enthalten von dem als Faden (‘thread’) bezeichneten sequentiellen Kontrollfluß gleichzeitig mehrere aktive Fäden, deren aktuelle Instruktionen zur Ausführung bereit stehen.

Die Verwendung mehrerer Fäden dient im wesentlichen dazu, vorhandene Verarbeitungseinheiten besser auszunutzen.

Wartezeiten von Verarbeitungseinheiten, die durch Abhängigkeiten innerhalb eines Fadens zwischen Einheiten entstehen, sollen durch die Fortführung der Aktivität eines anderen Fadens versteckt werden. Mehrfädige Architekturen sind speziell dafür entworfen, solche Latenzzeiten zu verstecken (‘latency hiding’). Dazu müssen sie in der Lage sein, den Kontrollfaden sehr schnell zu wechseln. Diese Eigenschaft macht sie als Knotenrechner von parallelen Systemen besonders interessant, da sie sowohl die Kommunikationslatenz als auch die Speicherzugriffslatenz verstecken können.



Thread (Kontrollfaden) or lightweight process is a strictly sequential thread of control (program code). It consists of a part of the program code and has:

- one entry
- one exit
  - local data
  - internal state

The strictly sequential order within a thread is often deemphasized, so that also loops and jumps back are allowed, but no jump directly into or out of the thread.

Threads share data! They use a common address space.

Multithread Architectures (MTA)

Mehrfädige Architekturen

Die Strategie, wann ein Fadenwechsel vorzunehmen ist, ist abhängig von den Operationen, deren Latenzzeiten versteckt werden sollen. Die folgenden Operationen mit fester oder variabler Latenzzeit können zu einem Fadenwechsel benutzt werden:

(1) Instruktionen

(2) Blöcke von Instruktionen

(3) nicht erfolgreiche Cachezugriffe

(4) entfernte Speicherzugriffe

Multithread Architectures (MTA)

Mehrfädige Architekturen

(1) Diese Strategie verschränkt die Verarbeitung von einzelnen Instruktionen aus mehreren Fäden und nutzt die Verarbeitungspipeline eines Prozessors sehr gut aus. Werden alle Instruktionen dieser Fadenwechselstrategie unterworfen, so können alle vorkommenden Latenzzeiten versteckt werden. Die aufeinanderfolgenden Instruktionen können in der Pipeline nicht zu einem Konflikt führen, da sie aus nterschiedlichen Fäden stammen und damit datenunabhängig sind. Die Kosten eines Fadenwechsels müssen bei solch feiner Granularität extrem gering sein, um bei deren großen Häufigkeit nicht zu einer Verringerung der Verarbeitungsleistung zu führen. Der Prozessor muß also Hardwareeinrichtungen zur Verarbeitung einer genügenden Anzahl von Fäden besitzen und auch deren Kontexte ständig für die Verarbeitungspipeline bereit halten.

(2) Blöcke von Instruktionen bewirken eine gröbere Granularität und reduzieren dadurch die Häufigkeit der Fadenwechsel. Die Kosten des Wechsels können durch das seltenere Auftreten höher sein und die erforderliche Hardwareunterstützung für mehrere Fäden kann einfacher ausfallen. Optimierungen der Länge der Blöcke können vom Compiler nach einer Datenabhängikkeitsanalyse vorgenommen werden, und die Lokalität bei der Verarbeitung eines Fadens wird besser ausgenutzt. Bei welcher Instruktion der Wechsel erfolgen soll, kann je nach Prozessortyp unterschiedlich sein.

(3) Wird ein MTA-Prozessor mit einem Cache ausgestattet, so sind erfolgreiche Cachezugriffe latenzarm (ein oder zwei Takte) und werden normalerweise von der Verarbeitungspipeline toleriert. Ist der Zugriff nicht erfolgreich (‘cache miss’), so kann das Nachladen der Cachezeile aus dem lokalen Speicher oder aber auch aus einem entfernten Speicher (‘remote memory access’) eines gemeinsamen Adreßraums erfolgen, was unter Umständen zu erheblichen und auch nicht vorhersagbaren Latenzzeiten führen kann. In einem solchen Fall wird der Faden gewechselt, um die Latenzzeit des lokalen oder des entfernten Speicherzugriffs zu verstecken.

(4) Kann der Compiler die entfernten Zugriffe bereits markieren, so können die Fäden auch nur bei solchen, mit sehr hoher und variabler Latenzzeit behafteten Instruktionen, gewechselt werden. Diese Variante ist von der Granularität schon so grob, daß die Hardwareunterstützung des Multithreading gering ausfallen kann, oder sogar ein Standardprozessor mit schnellem programmierten Kontextwechsel benutzt werden kann.

Bei allen diesen Strategien wird angenommen, daß weitere Fäden zur Verarbeitung bereit stehen. Ist das der Fall, so kann der Prozessor die Arbeit an einem anderen Faden fortsetzen, was die Ausnutzung verbessert.

Multithread Architectures (MTA)

Cost Evaluation

Bewertet man in einer Rechnerarchitektur die Kosten der verschiedenen Funktionseinheiten (‘functional units’ FU), so sollten die Einheiten mit den höchsten Kosten auch am besten ausgenutzt werden, d.h. sie sollten während des Auftretens von Blockierungen zur Verarbeitung weiterer aktiver Instruktionen anderer Fäden benutzt werden. Die Kostenfunktion ist natürlich abhängig von der verwendeten Technologie und der Komplexität der FU. Steht die Ausnutzung der arithmetischen Ausführungspipeline im Vordergrund, so können zur Vermeidung von Pipelineabhängigkeiten der Reihe nach Instruktionen von verschieden Fäden (‘instruction interleaving’) benutzt werden, da diese per Definition datenunabhängig sind. Sind mindestens soviele verschieden Fäden verfügbar, wie die Pipeline Stufen besitzt, so kann sogar auf eine Datenabhängigkeitserkennung und -steuerung der Pipeline verzichtet werden.

The overall goal is to implement a chosen program execution model in a way that offers high performance for the least cost.

When do I get the highest performance?

- at the maximum utilization of all resources

When do I have the lowest costs?

- using the cheapest technology and system design

Example calculation for FP Unit cost

PowerPC 620
1600,- DM/die 311 mm^2

$14 * 15 = 210$ relative die size of PowerPC 620 measured values of ’die’ photo

$210 \triangleq 311\ mm^2$ $\frac{311}{210} \approx 1{,}48$ factor for conversion to mm^2

$5{,}5 * 5 = 27{,}5$ relative die size of FP Unit

$27{,}5 * 1{,}48 = 40{,}7\ mm^2$ die size of FP Unit

$\frac{40{,}7\ mm^2}{311\ mm^2} = 0{,}13$ area fraction of FP Unit

$0{,}13 * 1600{,}- = 209{,}-DM$ cost of FP Unit

Multithread Architectures (MTA)

Mehrfädige Architekturen

Eine der ersten MTA mit einer solchen Pipelinestruktur war der Heterogenous Element Processor (HEP) [Burton Smith]. Die folgende ausführliche Beschreibung seiner Architektur dient dem Zweck, die Strategien (1) und (4) mit ihren erforderlichen Hardwareeinrichtungen aufzuzeigen. Der HEP kann gleichzeitig 16 Tasks mit jeweils 128 aktiven Kontrollfäden verwalten und besitzt spezielle Hardware zur Kontrollfadenerzeugung und Terminierung. Der Prozessor bildet die Ausführungseinheit (‘process execution module’ PEM) des HEP-Parallelrechner. Das Parallelrechnersystem ist als eine Architektur mit gemeinsamem Speicher konzipiert. Um die Zugriffslatenzen zu diesem gemeinsamen Speicher zu verstecken, werden die Lade- und Speicherbefehle vom Scheduler einer Speicherzugriffseinheit (‘storage function unit’ SFU) übergeben, die ebenfalls mehrere Instruktionen unterschiedlicher Fäden gleichzeitig verwalten kann. Ist die Speicheroperation erfolgt, so wird der Prozeßbezeichner PT dieses Kontrollfadens wieder zurück an die Kontrollschleife gesendet und kann damit fortgesetzt werden. Für dieses Verfahren, die Latenzzeit des Speicherzugriffs zu verstecken, wurde die Bezeichnung ‘split-phase transaction’ geprägt.

Jede Task hat ihre eigenen geschützten Program-, Daten- und Registerbereiche, die durch das Task-Statusword (‘task status word’ TSW) beschrieben werden. Die TSW für diese 16 schwergewichtigen Prozesse werden ständig im Prozessor gehalten und bei jeder Ausführung einer Instruktion zur Berechnung der aktuellen Adressen von Datenobjekten herangezogen. Die 16 Task-Queues enthalten die jeweils einer Task zugeordneten 64 möglichen Einträge von Kontrollfädenbezeichnern (‘process tag’ PT) . Die 7 bit des PT zeigen auf 128 mögliche Prozeßstatusworte (‘process status word’ PSW), die die leichtgewichtigen Prozesse (‘threads’) beschreiben. Das PSW beschreibt den aktuellen Zustand eines Kontrollfadens und enthält den Instruktionszeiger, einen 4 bit Taskidentifizierer und die Offsets für die Adressierung der Registersätze.

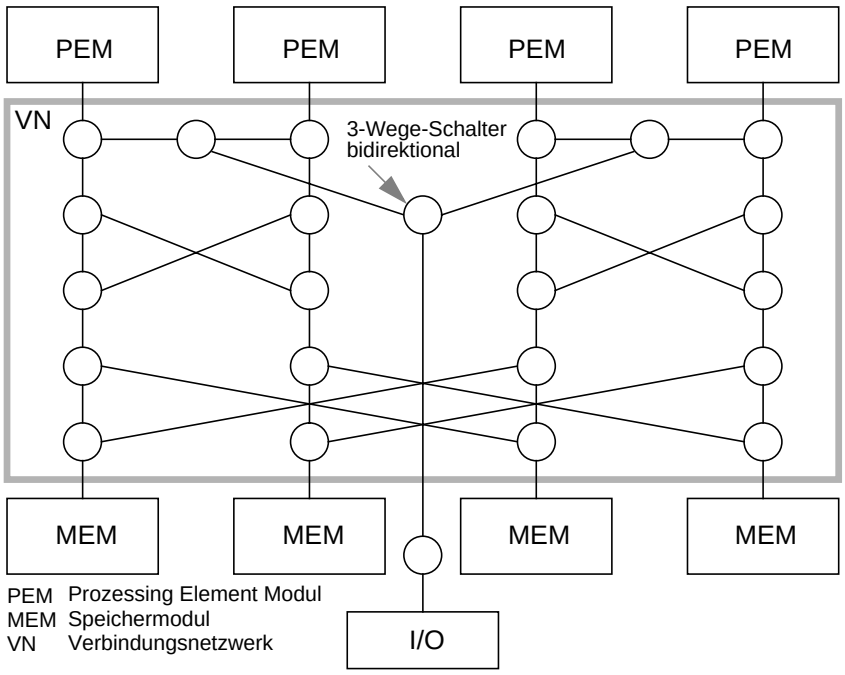
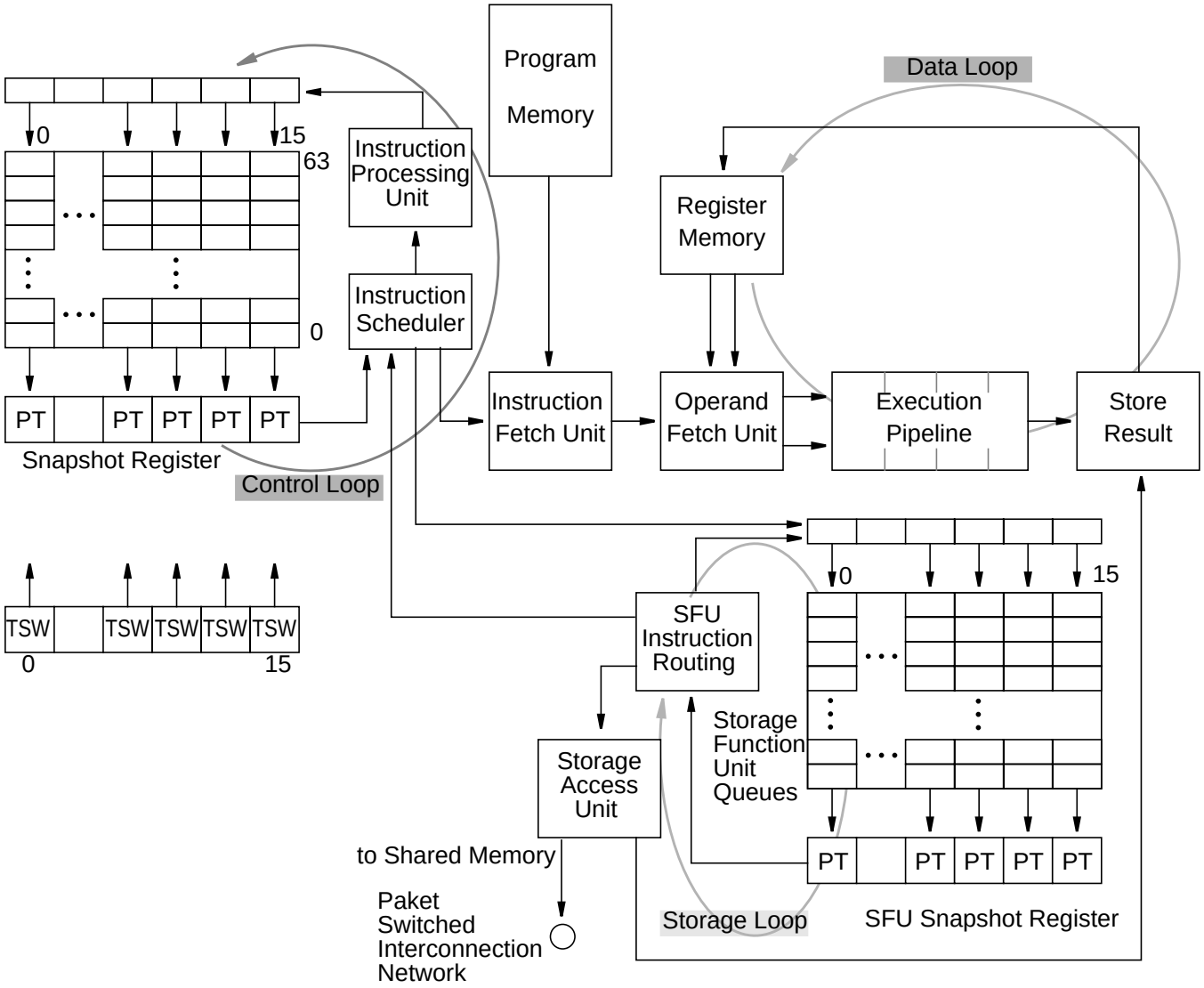


Abb. 1.11 Architektur des HEP-Parallelrechners mit 4 Prozessoren

Multithread Architectures (MTA)

Multithread Architectures (MTA)

Die Struktur eines Prozessors (‘processing element module’ PEM) ist in der folgenden Abbildung vereinfacht dargestellt.



Struktur der Ausführungseinheit des PEM

In der Kontrollschleife (‘control loop’) zirkulieren die PT, so daß eine erneute Ausführung desselben Fadens erst nach 8 Takten der Ausführungspipeline erfolgen kann. Das PT-Register entnimmt jeweils 16 PT aus den Task-Warteschlangen und gibt sie sequentiell an die Verarbeitungseinheit weiter. Dadurch wird die Verarbeitungseinheit den Tasks immer fair zugeteilt und die Instruktionen der Tasks verschachtelt (‘instruction interleaving’). In der Datenschleife werden die vom PSW-Offset und der Instruktion adressierten Operanden aus dem Registerspeicher gelesen, verarbeitet und zurückgespeichert (innerhalb von 8 Takten).

Multithread Architectures (MTA)

Multithread Architectures (MTA)

Der Registerspeicher enthält die Kontexte (Registersätze) aller im Prozessor geladenen Fäden. Für die externen Speicherzugriffe wurde ebenfalls eine Task-Warteschlage verwendet (‘storage loop’). Der PT für einen Speicherzugriff wird in die Speicher-Task-Warteschlage eingereiht und nach Beendigung der Speicheroperation wieder zurück in die Kontrollschleife gegeben. Solange der PT in der Speicher-Task-Warteschlage ist, ist dieser Faden von der Ausführung suspendiert.

Mit der Architektur des HEP-Rechner wurde versucht, die zu dem Zeitpunkt seiner Realisierung recht teure arithmetische Verarbeitungspipeline so effizient wie möglich zu nutzen und die externen Speicherzugriffe zu verstecken. Der wesentliche Nachteil des HEP war die starke Leistungsreduzierung, wenn nur ein Faden zur Verfügung steht, denn dann sinkt die Leistung des Prozessors auf 1/8 (nur jeder 8.Takt der Verarbeitungseinheit kann von selben Faden benutzt werden.

Eine Neuentwicklung, die auf ähnlichen Mechanismen des Multithreading beruht, ist das TERA-System, welches in [Tera] näher beschrieben ist.

Der APRIL-Prozessor [Aga90] des ALEWIFE-Systems ist ebenfalls eine mehrfädige Architektur, nutzt aber die Strategie (2) zum Fadenwechsel. Er basiert auf einer geringfügigen Modifikation des SPARC-Prozessors, dessen überlappendes Registerfile durch Softwarekontrolle zu mehreren festen Registersätzen für 4 Kontrollfäden umstrukturiert wurde. Für jeden Faden sind zwei Registersätze bestimmt, einer für den Benutzer und einer für die schnelle Ausnahmebehandlung (‘trap’). Ein Fadenwechsel zwischen diesen vier geladenen Fäden (‘loaded threads’) erfordert nur das Retten des Programmzählers und des Prozessorstatusregisters in den faden-eigenen Ausnahmeregistersatz und kostet damit nur ca. 4 bis 10 Takte. Für den Wechsel sind keine externen Speicherzugriffe erforderlich. Da nur vier Fadenkontexte im Prozessor geladen sein können, werden die anderen nicht geladenen Fäden mittles einer Warteschlange im Hauptspeicher gehalten. Sind alle vier geladenen Fäden nicht ausführungsbereit, so muß ein geladener Faden mit einem im Speicher liegenden, ausführungsbereiten Faden gewechselt werden. Erst bei dieser Operation erfolgt die Auslagerung des Kontextes eines Fadens in den Hauptspeicher.

Im MANNA-Parallelrechnersystem [Manna] werden zwei Fäden gleichzeitig durch zwei vollständige superskalare Prozessoren im Knoten realisiert. Sollte ein Faden blockieren, so kann dieser Prozessor während dieser auftretenden Latenzzeit einfach ungenutzt bleiben, da die Prozessoren innerhalb eines Knotens sehr kostengünstig repliziert werden können. Zukünftige Knotenarchitekturen werden von diesem einfachen und kostengünstigen Ansatz verstärkt Gebrauch machen, um die Vorteile der mehrfädigen Verarbeitung nutzen zu können.

Die Vorteile von MTA bestehen allgemein in der guten Ausnutzung der Verarbeitungsleistung und der Toleranz bezüglich der verschiedenen Latenzzeiten eines Parallelrechnersystems.

Multithread Architectures (MTA)

Multithread Architectures (MTA)

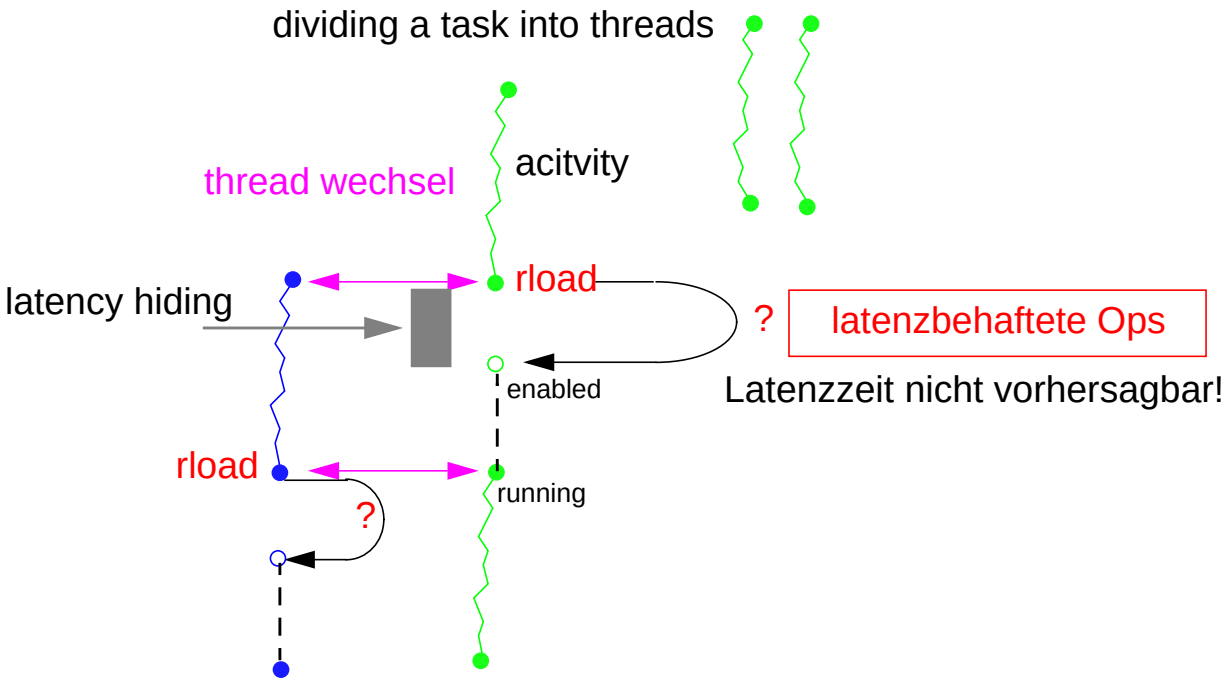
Single thread vs. multiple threads

Design space questions:

Which functions are the most expensive ones?

What kind of latency shall be hidden?

Which unit is underutilized due to latency?



Goal: increase of HW (Processor) Utilization

Multithread Architectures (MTA)

Multithread Architectures (MTA)

Multiple Context Processors

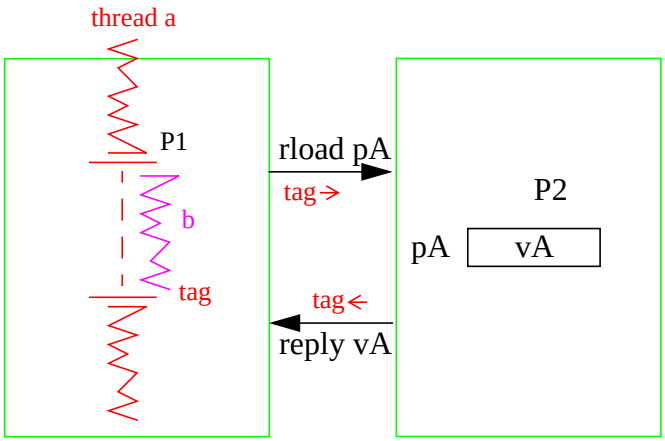
- latency hiding by fast switch to next thread within one processor
- latency hiding using node with multiple CPUs

(Context) switch policies

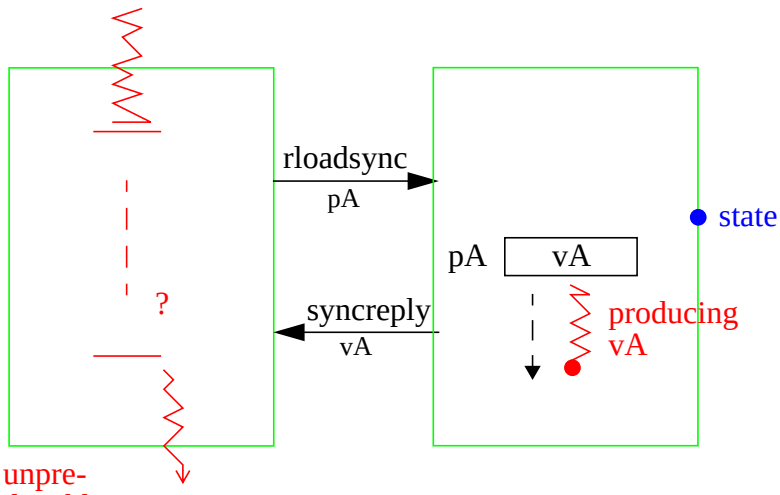
thread

- switch on every instruction HEP
  interleaving of instructions from different threads on a cycle by cycle basis
- switch on every load
  interleaving threads to hide memory access latency
  better: switch on a cache miss! APRIL
- switch on a block of instructions
  interleaving of blocks from different threads

- remote load



- synchronizing load



- complete time
- scheduling

synchronization at full/empty

Multithread Architectures (MTA)

Multithread Architectures (MTA)