Analysis and Modeling of the Performance of LQCD Kernels

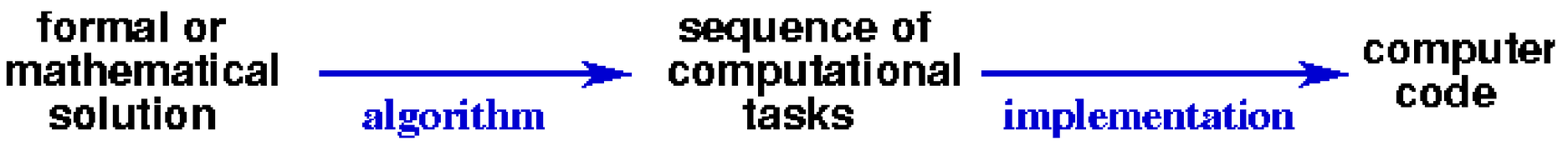
Hubert Simma

DESY

Outline:

- ❑ Motivation
- ❑ Rough Estimates
- ❑ Simple Performance Modeling
- ❑ Hardware Model
- ❑ Analysis of Computational Tasks
- ❑ Wilson-Dirac Operator

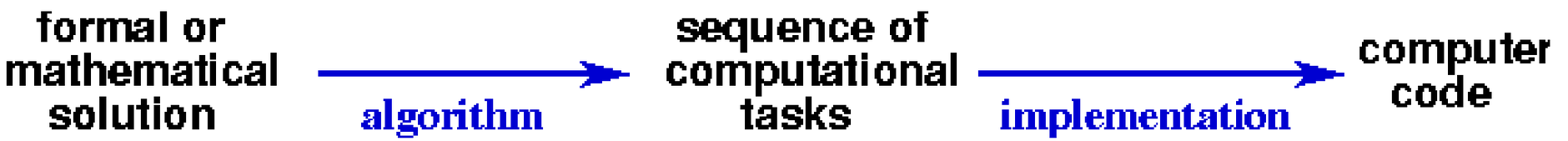
Algorithm vs. Implementation



Computational tasks:

- different levels of granularity, e.g.
  - update U
  - compute plaquettes or MD forces
  - solve $Ax = y$ or Dirac matrix $\times$ vector
  - basic linear algebra, random numbers, . . .
- often described by pseudo-code
- NO reference to specific programming language or machine architecture

Algorithm vs. Implementation



Implementation:

- high-level coding (human)
  - data representation
  - memory layout
  - selection and scheduling of (macro-)operations
  - management of communications
- low-level code generation (compiler)
  - management of memory accesses
  - register allocation
  - selection and scheduling of (micro-)instructions

Why benchmarking and modeling?

May want to improve or may not (yet) have available

- algorithm
- application code (high-level implementation)
- system software (low-level implementation)
- hardware

Key Question:

> How much wall-clock time is (expected to be) required to solve a given task on a given architecture?

$$\mathbf{T} = \mathbf{f}\Big(\underset{\text{task}}{\underbrace{\text{algorithm}}} \;,\; \underset{\text{code}}{\underbrace{\text{implementation}}} \;,\; \text{machine} \Big)$$

Run-time profiling

routine	calls	time
Dirac operator (3 variants)	80844	58 %
Linear algebra (3 routines)	60736	26 %
Gauge forces + update	320	8 %
Global sum ($4 \times 8 \times 8$ nodes, 128 bit)	83554	0.4 %
Others ($\approx$ 70 routines)		7 %

provides

- algorithmic cost, i.e. number of computational tasks
- CPU cost, i.e. time for computational tasks
- hints for optimisations (of algorithm, implementation, or machine)

Application signatures

Total Computing Cost

$$N_{ops} \equiv \#\ \text{FP operations}$$

Computing vs. Memory Access

$$R_{ops} \equiv \frac{\#\ \text{arithmetic operations}}{\#\ \text{memory accessess}}$$

Communication Requirement

$$R_{rem} \equiv \frac{\#\ \text{remote accesses}}{\#\ \text{memory accessess}}$$

Examples of LQCD tasks (kernels):

kernel	N_{ops} flop/site	R_{ops} flop/cword	R_{rem}
linear combination $a \cdot \underline{x} + \underline{y}$	8	2.6	0
dot product $(\underline{x}, \underline{y})$	8	4	$2G/V$
norm $\|\|x\|\|^2$	4	4	$1G/V$
$SU(N)$ matrix $\times$ vector	$N(8N-2)$	$\frac{8N-2}{N+2}$	0
Dirac operator $D\phi$	1320	≥ 7	A/V

- $V = L_x \times L_y \times L_z \times L_t$ is the 4-d lattice volume
- $A = \sum A_i$ or $A = \max A_i$ (depending on network architecture) and A_i is the number of surface sites in direction i
- $G =$ communications to perform a global sum (with or without broadcast of result)

➠ refined analysis . . .

Hardware characteristics

Memory System

$$\rho_{mem} \equiv \frac{\text{flops}}{\text{bandwidth}} \quad [flop/byte]$$

Communication Network

$$\rho_{net} \equiv \frac{\text{network bandwidth}}{\text{memory bandwidth}}$$

Balance: Application vs. Hardware

$$R_{ops} \approx \rho_{mem} \qquad \text{and} \qquad R_{rem} \approx \rho_{net}$$

➠ refined model . . .

A simple Performance Model

Execution time

Consider (micro-)tasks μ of different devices,
e.g. arithmetic units, memory ports, communications, . . .

$$T_{exec} \geq \max_{\mu} T_{\mu}$$

- assumes concurrency of different tasks (otherwise split tasks)
- ignores data dependencies between tasks
  (assume optimised software pipelining/prefetch)
- may include or neglect pipeline latencies

Upper bound (sequential execution)

$$T_{exec} \leq \sum_{\mu} T_{\mu}$$

Efficiency

$$\epsilon \equiv \frac{T_{peak}}{T_{exec}} \quad \text{with} \quad T_{peak} = N_{flop}/\beta_{FP}$$

A simple Hardware Model

What describes a computer architecture?

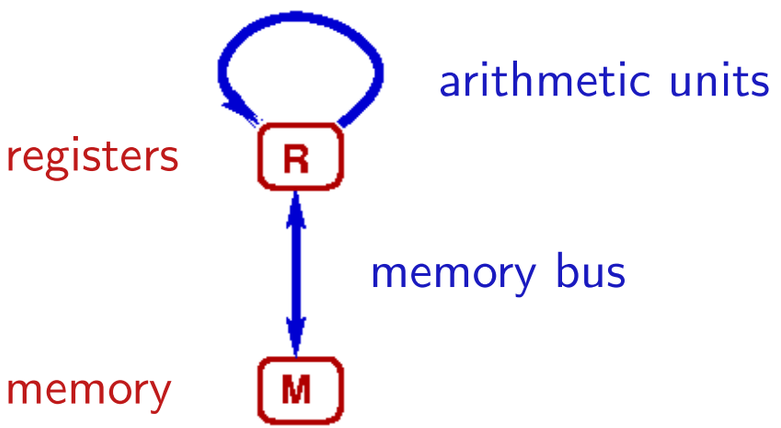
Set of hardware devices/units for:

- control (of data and program flow)
- storage of data (and code)
  - memory
  - cache(s)
  - registers
  - buffers, fifos, flip-flops, ...
- processing/transport (of data)
  - arithmetic operations (usually pipelined)
  - storage access (hopefully pipelined)
  - combinatorical logics
  - buses

Hardware structure:

i.e. a graph with

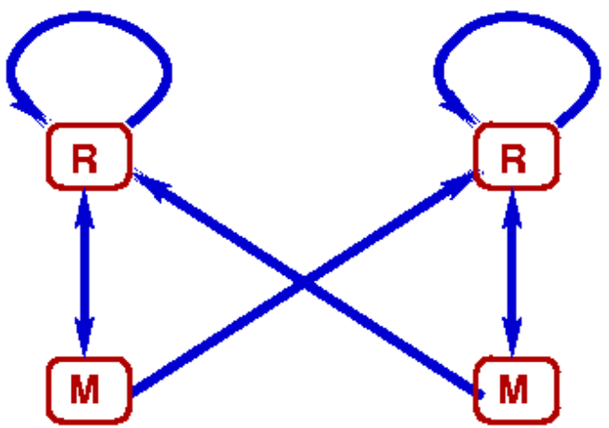
- vertices = storage devices
- edges = data paths through processing/transport devices



Hardware structure:

i.e. a graph with

- vertices = storage devices
- edges = data paths through processing/transport devices

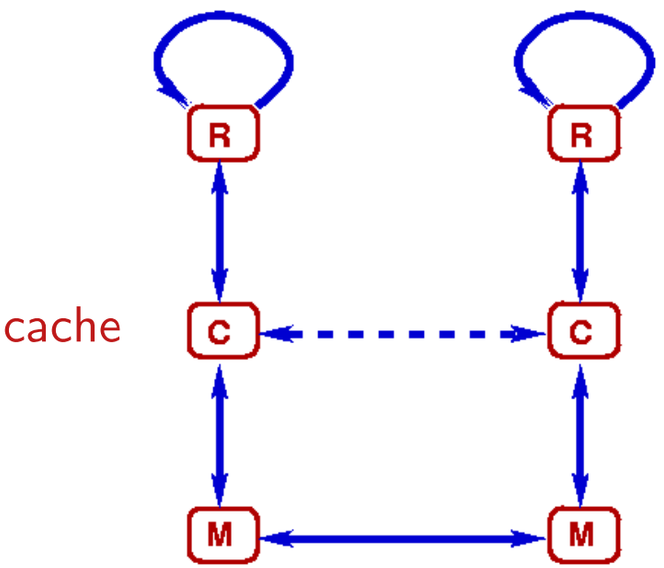


network

Hardware structure:

i.e. a graph with

- vertices = storage devices
- edges = data paths through processing/transport devices



What are the hardware parameters?

(to be determined from data sheets or micro-benchmarks)

Storage devices:

σ_i = storage size

Data units: bit, byte, fword, cword, . . .

Processing/transport devices:

ISA = instruction set architecture

β_i = bandwidth (data throughput/time)

λ_i = latency (delay between input and first output)

Time units: nsec, T_{clk} (clock cycle)

N.B.: For storage access one might need: $\lambda = \lambda_0 + \delta(a, a', n, n')$

Hardware examples:

parameter	unit	PC	BG/L	APE
f_{clk}	[GHz]	2	0.7	0.14
data format	[1 word]	s	d	d
β_{RR}	[flop/clk]	4 s	8 d	8 d
σ_R	[word]	≤ 100	64	512
cache		L2	L3	—
β_{RC}	[word/clk]	4	2	—
σ_C	[word]	0.5 M	0.5 M	—
β_{CM}	[word/clk]	1/8	1	2
λ_{CM}	[clk]	≥ 100	≥ 30	≈ 20
$\beta_{PP'}$	[word/clk]	0.1	0.03×16	0.1×12
$\lambda_{PP'}$	[clk]	≥ 2000	≈ 700	≈ 40

Analysis of Computational Tasks (and their Implementation)

Instruction Match

Matching between required operations and machine instructions (ISA)

- single vs. double precision
- real vs. complex
- multiply/add vs. MulAdd
- alignment constraints (of vector or "SIMD" instructions)
- . . .

Example: ISA has only complex MulAdd (possibly with complex conjugation)

kernel	flop needed	MulAdd used	ϵ_{max}
sum of real numbers	N	N	12.5 %
linear combination $a \cdot \underline{x} + \underline{y}$	$N \times 8$	N	100 %
dot product $(\underline{x}, \underline{y})$	$N \times 8 - 2$	N	$\approx$100 %
norm $\|\|x\|\|^2$	$N \times 4$	N	50 %
Dirac operator $D\phi$	166×8	276	60 %

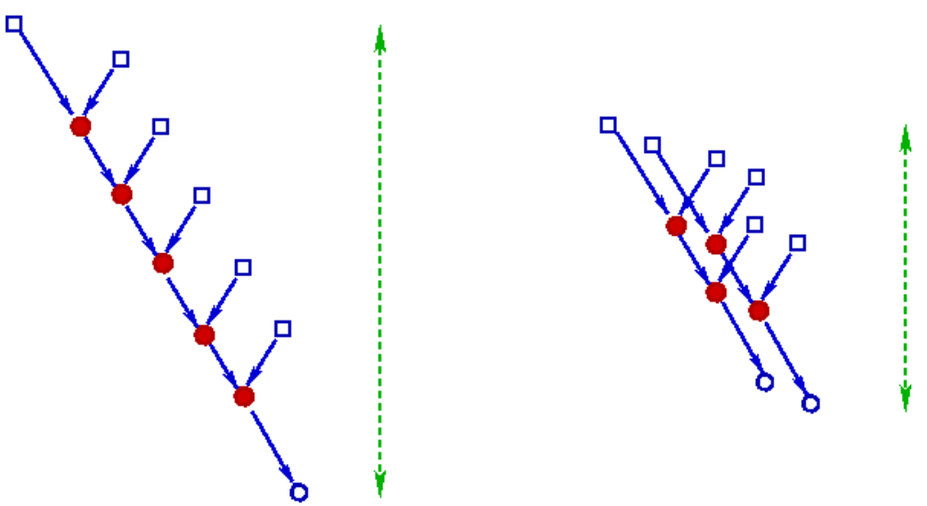
Data Flow

Consider dependency graph of instructions (DAG = directed acyclic graph)

- vertices = (groups of) instructions
- edges = (intermediate) data ⇒ partial order

Example:

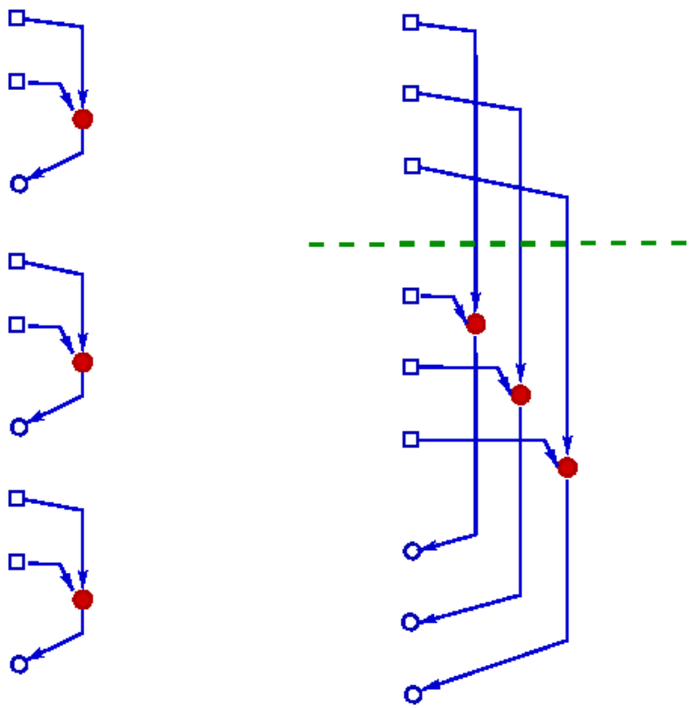
$$s = \sum_{i=0}^{N} x_i \quad \rightarrow \quad s_n = \sum_{i=0}^{N/K} x_{nK+i}$$



☆ Critical path ≃ "height" ⇒ minimal latency-limited execution time

Example:

$$\underline{z} = \underline{x} + \underline{y}$$

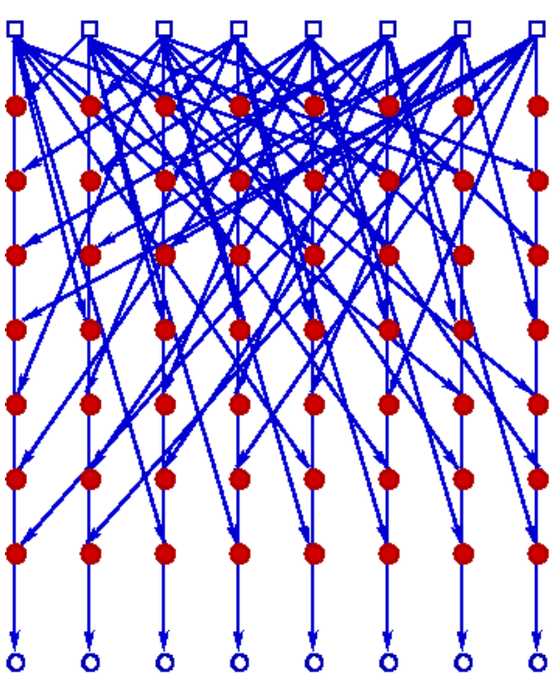


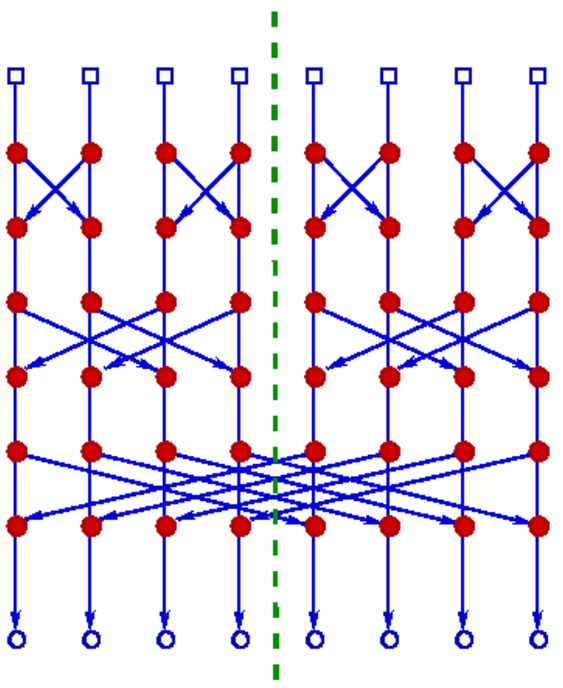
"causal cuts" = possible time-order (scheduling)

☆ Flow accross cut ⇒ storage requirement

Example:

$$DFT \quad \rightarrow \quad FFT$$





arbitrary cuts = possible parallelisations

☆ Flow accross cut ⇒ communication requirement

Information Exchange

$I_{XY}(N, \sigma) \equiv$	data exchange for specific computational task of size N between computer sub-systems X and Y with storage σ_X

where X, Y = registers (R), memory (M), cache (C), processors (P, P'), ...

More explicit: For one or more implementations compute separately

- $I_{XY}(N) \Rightarrow$ bandwidth requirements
- $S_X(N) \Rightarrow$ storage requirements

Execution time estimate:

$$T_{XY} \approx I_{XY} \cdot \beta_{XY} + O(\lambda_{XY})$$

(if $S_X < \sigma_X$, otherwise split tasks)

N.B.:

$$N_{ops} \sim I_{RR} \ , \quad R_{ops} = \frac{I_{RR}}{I_{RM}} \ , \quad R_{rem} = \frac{I_{PP'}}{I_{RM}}$$

Optimisation Strategies and Tradeoffs

Try to combine, re-arrange or modify computational tasks
to reduce data flow through time-critical paths

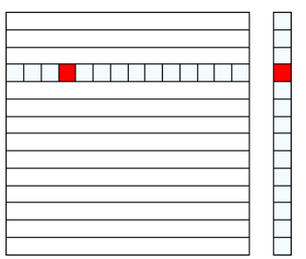
$\Rightarrow$ typical conflicts:

- S_R vs. I_{RC} (I/O overhead)
- S_C vs. I_{CM} (cache misses)
- S_M vs. $I_{PP'}$ (communication overhead)

Example: matrix × vector

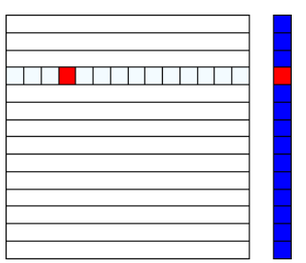
$$y_i = \sum_j M_{ij} v_j \quad (i, j = 1, \ldots, N)$$

Implementation without cache



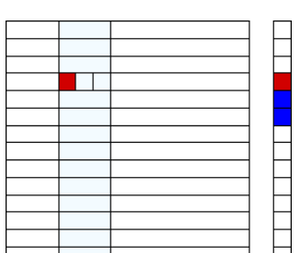
$$\begin{aligned} S_C &= 0 \\ I_{RM} &= 2N^2 + N \end{aligned}$$

Optimally cached implementation



$$\begin{aligned} S_C &\leq 1 \ldots 2N \\ I_{CM} &= N^2 + 2N \end{aligned}$$

Block-wise implementation

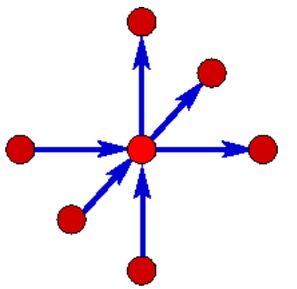


$$\begin{aligned} S_C &\leq 2B \\ I_{CM} &= N^2 + 2N + \left(\tfrac{N}{B} - 1\right) \cdot 2N \end{aligned}$$

Analysis of the Wilson-Dirac Operator

Hopping term:

$$[D\phi]_x \equiv \sum_{\mu=1}^{4} \{U(x,\mu)(1-\gamma_\mu)\phi(x+\hat{\mu}) + \cdots\}$$



Implementation without cache

$$\begin{aligned} S_C &= 0 \\ I_{RM}/v &= (8+1)|\phi| + 8|U| = 180\ cword \end{aligned}$$

(v = number of lattice sites, $|\phi|$ = size of ϕ per site)

Optimally cached implementation

$$\begin{aligned} S_C/v &= 1|\phi| + 4|U| &&= 48\ cword \\ I_{CM}/v &= 2|\phi| + 4|U| &&= 60\ cword \\ I_{RC}/v &= (8+1)|\phi| + 8|U| &&= 180\ cword \end{aligned}$$

Partially cached implementation

- Work on 4-d sub-lattices with $v' \equiv l'_x \times l'_y \times l'_z \times l'_t$ sites and

$$a'_+ \equiv v' \sum_i \frac{1}{l'_i}$$

  sites on surfaces in positive direction

- Holding all sites of sub-lattice in cache requires

$$\begin{aligned} S_C/v' &\geq 1|\phi| + 4|U| \\ I_{CM}/v &= 2|\phi| + 4|U| + (2|\phi| + |U|) \cdot a'_+/v' \end{aligned}$$

- For minimal surface a'_+/v' choose $l'_x = l'_y = l'_z = l'_t \quad (\Rightarrow a'_+/v' = 4/l'_t)$

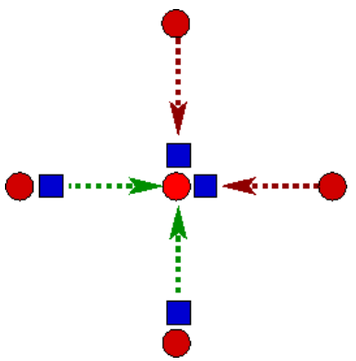
Parallel implementation

Local lattice on each node:

$$v \equiv l_x \times l_y \times l_z \times l_t$$

Sites on positive surfaces of local lattice:

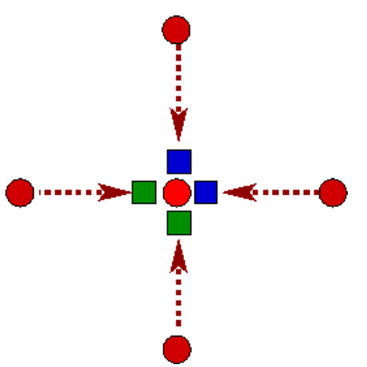
$$a_+ \equiv v \sum_{k:l_k \neq L_k} \frac{1}{l_k}$$



$$I_{PP'} = (2|\phi| + |U|) \cdot 2a_+ \;=\; 66 \cdot a_+ \; cword$$

Pre-distributed U

Prepare auxiliary array $U'(x,\mu) \leftarrow U(x+\hat{\mu},\mu)$
to guarantee that all U fields are available locally when needed

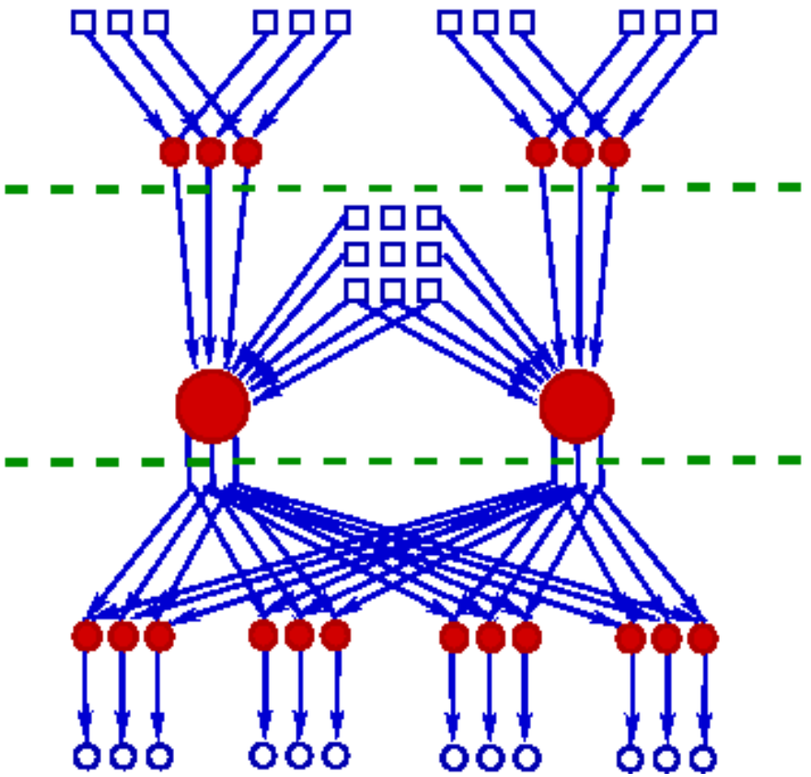


$$I_{PP'} = \left(2|\phi| + O(\frac{1}{N_{it}})|U|\right) \cdot 2a_+ \;\approx\; 48 \cdot a_+ \; cword$$

but increased storage requirement $S_M/v = N_\phi|\phi| + (4+d) \cdot |U|$

Intermediate 2-spinors

DAG of hopping term in one direction:

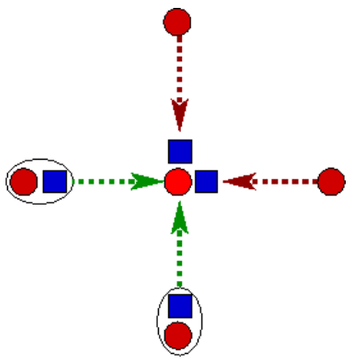


- Projection on 2-spinors:

$$p_\mu^\pm(x) = (1 \mp \gamma_\mu) \cdot \phi(x)$$

- Multiplication with pre-distributed U before or after communication

$$q_\mu^\pm(x) = U(\ldots,\mu) \cdot p_\mu^\pm(x \pm \hat{\mu})$$



- Reconstruction of 4-spinor:

$$\phi'(x) = \sum_\mu \Big\{ R_\mu^+\big(q_\mu^+(x)\big) + R_\mu^-\big(q_\mu^-(x)\big) \Big\}$$

$\Rightarrow$ reduced communications

$$I_{PP'} = \Big(1|\phi| + O(\frac{1}{N_{it}})|U|\Big) \cdot 2a_+ \;\approx\; 24 \cdot a_+ \; cword$$

but increased register I/O to cache (or memory)

$$\begin{array}{lcll} I_{RC}/v & = & (1+\frac{8}{2}+\frac{8}{2}+1)|\phi| + 8|U| & = \quad 192 \; cword \\ S_C/v & \geq & \frac{8}{2}|\phi| & = \quad 48 \; cword \end{array}$$

Examples at algorithmic level

Iterative solvers:

Combine differnt point-operations while data in registers/cache

Example:

- $s \leftarrow r + \beta \cdot s$
- $q \leftarrow A \cdot s$
- global (s, q)
- set $\alpha \leftarrow \frac{(r,r)}{(s,q)}$

- compute locally

$$\begin{array}{lll} s & \leftarrow & r + \beta \cdot s \\ q & \leftarrow & A \cdot s \\ (s,q)_{loc} & & \end{array}$$

- compute global (s, q) and set

$$\alpha \leftarrow \frac{(r,r)}{(s,q)}$$

$I_{CM}/v = 7|\phi|$  $I_{CM}/v = 4|\phi|$

Even-odd peconditioning:

- $I_{CM}, S_C, S_M \sim N_\phi|\phi| + N_U|U| \rightarrow \frac{N_\phi}{2}|\phi| + N_U|U|$
- might keep some U in registers/cache for $D_{eo}D_{oe}$
- might work on 5 time slices

Schwarz Alternating Procedure:

- natural decomposition into cache-friendly domains
- $I_{PP'} \rightarrow \frac{1}{N_{MR}} I_{PP'}$

Summary

Simple methodology for performance modeling based on

- general hardware model parametrized by
  - storage sizes
  - bandwidths
  - latencies
- analysis of computational tasks in terms of
  - instruction and storage requirements
  - data dependencies between operations
  - information exchange between hardware sub-systems

. . . can be applied at different levels accuracy

. . . may help to select/improve
  - algorithms
  - implementations
  - system software and hardware