Solving the Dirac equation on QPACE

Andrea Nobile

Uni Regensburg

OUTLINE

1. CELL PROCESSOR
2. SOLVER CHOICE
3. SAP GCR SOLVER
4. SAP-FGMRES-DR
5. CONCLUSIONS

Cell Architechture: Explicit handling of memory hierarchy

- 1 standard core (PPU), 8 small cores (SPU)
- 256KB Local Store per SPU it is not a cache
- In order to access data in main memory (MM) from the SPU, the programmer must give explicit commands to a dma machine to move data between LS and MM
- SPU program works only on LS addresses
- Data layout must be carefully chosen both to facilitate programming and to obtain performance
- Tasks with irregular/complex data access pattern are not well suited for SPU

Solver choice

	CG	SAP-GCR (M. Luescher)
Lattice size flexibility	poor	reasonable
E/O preconditioning	difficult	yes (block solver)
Network bandwidth req.	high	low
Network latency tolerance	moderate	high
MM bandwidth req.	moderate/high	moderate

- Adaptation for BQCD (Y. Nakamura)
- BQCD
  - Y. Nakamura
  - H. Stueben

SAP GCR Solver overview

- FGCR algorithm
- Domain Decomposition Preconditioner (SAP)
- Mixed precision (restart done in DP)

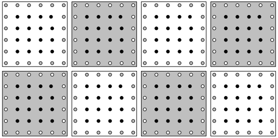
Using M_{sap} to denote the operator associated with the SAP the righ-preconditioned Dirac equation reads

$$DM_{sap}\phi = \eta$$

the solution ψ of the original equation $\psi = D^{-1}\eta$ is given by

$$\psi = M_{sap}\phi$$

PRECONDITIONER: SAP



- Lattice is divided into non overlapping blocks, chessboard coloured
- The two domains are visited alternatively to construct an approximate solution $\psi = M_{sap}\rho$ of the equation $D\psi = \rho$
- For each SAP cycle and domain, once $D_\Omega \psi_\Omega = \rho_\Omega$ is solved on the block, the contribution to the residue on neighbouring blocks due to the updated solution is computed $\rho \to \rho - D\psi$
- The update of the residue requires network communication
- Data communicated through network is necessary only after half of the blocks are processed (ability to tolerate high network latency)

SAP Implementation

- Required a complete rewrite
- Block solver uses an MR algorithm with a fixed number of iterations n_{it} (4 - 16)
- Block solver is parallelized on chip along the time direction on the 8 SPUs
- Network bandwith requirements are $1/(n_{it}+1)$ of the equivalent Dirac operator
- To maximize block size and block solver performance, I have chosen to limit prefetch
- Network data transfer is overlapped with computation

TNW: A low level programmer's view

- 6 links, 8 channels (up to 8 concurrent messages on the same link)
- Send Operation
  - links and channels are memory mapped address ranges in the program address space
  - Sends operations are done with SPU's dma engine (mfc put operation) to the address of the wanted link and channel
- Receive Operation
  - Network Processor (NWP) must be programmed to enable data forwarding from a specific link-channel combination to a given address
  - NWP notifies the program for a completed recv operation by updating a user defined location in memory
  - Possible to receive data both in local store and main memory

SAP Implementation (Network)

- Network sends are done from SPUs using the SPU dma engine
- Recv operations are controlled by PPU and done directly in MM
- One (64KB) page per block face
- The number of recv operations in-flight is tunable from 1 up to 8 blocks per link (Network channels)
- Different SPUs use different remote offsets for puts (8 sends for each recv)
- SPU code identical to single-node version

SAP Performance

- Block solver performance limited by
  - LS size, code size
  - load/store/shuffle instructions
  - dma commands overhead
- block solver alone achieved 50% of the floating point peak on a $8^2 \times 6^2$ block (xlc)
- production block solver (gcc) runs at 36% on $8^2 \times 4^2$ block
- complete M_{sap} (with dmas, tnw comm., D on block boundaries, etc.) runs slower
- simple but good estimate of the M_{sap} performance can be done with

$$\epsilon_{fp} = \frac{T_{bs-peak} \times (n_{it}+1)/(n_{it})}{\max[T_{bs} \times (n_{it}+1)/(n_{it}) + T_{LM}/\epsilon_{LM}\ ,\ T_{tnw}]}$$

SAP PERFORMANCE

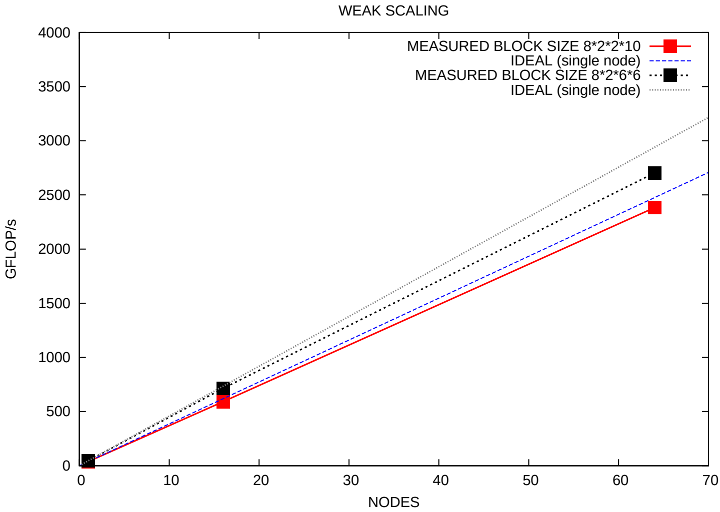
Typical solver parmaters: block solver iterations = 4, SAP cycles = 10

block size	ϵ_{bs} (m)	ϵ_{Msap} (e)	ϵ_{Msap} (m)	T_{fp}	T_{LM}	T_{tnw}(1.5)	T_{tnw}(3.0)
8x4x8x4	36%	25.8%	25.9%	458	145	128	64
8x2x6x6	34%	24.1%	23%	258	89	128	64
8x2x2x10	30%	21.7%	19.3%	172	52	94	47

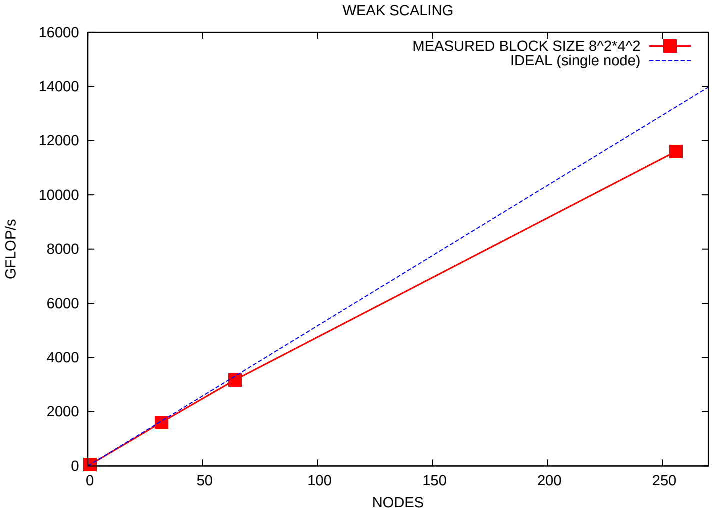
Typical production run on 32^3x64 lattice with $\kappa = 0.13632, \beta = 5.29$ on 256 QPACE nodes (2048 cores)

	gcr	Msap	dp D	la	cl	cl inv
fraction of total solver time	96%	72%	13%	12%	3%	$< 1\%$

SAP Scaling

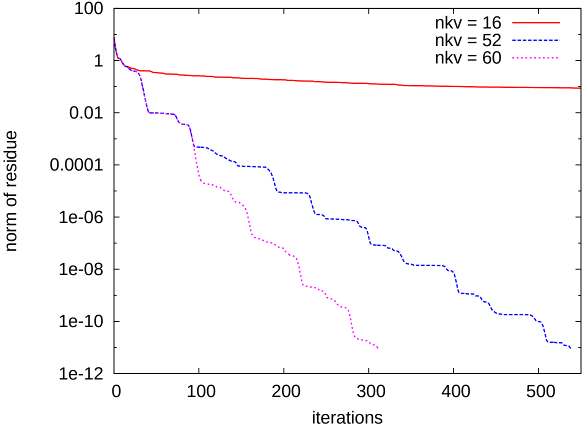

WEAK SCALING
MEASURED BLOCK SIZE 8*2*2*10
IDEAL (single node)
MEASURED BLOCK SIZE 8*2*6*6
IDEAL (single node)
GFLOP/s
4000
3500
3000
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
70
NODES

SAP Scaling



WEAK SCALING
MEASURED BLOCK SIZE 8^2*4^2
IDEAL (single node)
GFLOP/s
NODES
0
2000
4000
6000
8000
10000
12000
14000
16000
0
50
100
150
200
250

WHEN k GOES HIGH



$16^3 \times 32$, $\beta = 5.29$, $\kappa_{sea} = 0.1350$, $\kappa_{val} = 0.1374$, $\kappa^c_{val} \approx 0.1377$

The restart of death

- Restarting the recursion destroys the Krilov subspace
- Accumulated information is lost
- Information is reconstructed after every restart
- Recursion is too short to take advantage of accumulated information

What happens?

- SAP preconditioner
  - able to invert effeciently high modes
  - not able to invert low modes
- Let us speculate about what happens
  - At the beginning, the source has good overlap with high and low eigenvectors
  - After some iterations, maybe 1-2 restarts, high eigenvectors are $\approx$ absent in the residual vector
  - Residual vector has good overlap with the space of low eigenvectors
  - Under these conditions SAP is not an efficient inverter
  - Recursion is constructing a basis for a subspace spanned mostly by low eigenvectors
  - after this, solver proceeds as a deflated solver: fast

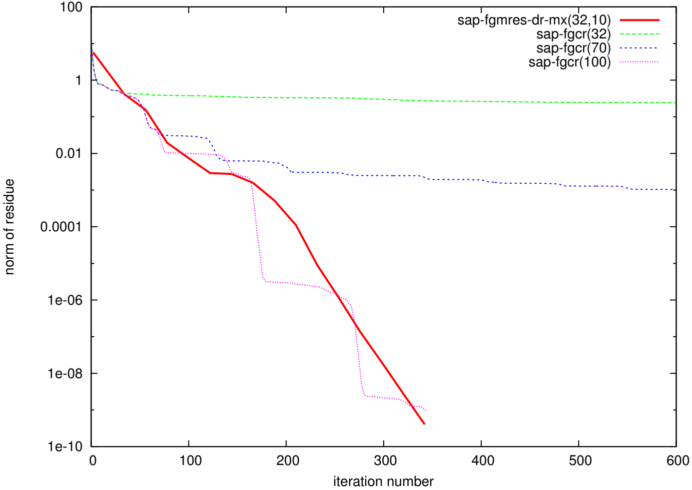
SAP-FGMRES-DR

- GMRES-DR (R. B. Morgan)
- FGMRES-DR (*F* generalization by X. Pinel)

Basic IDEA

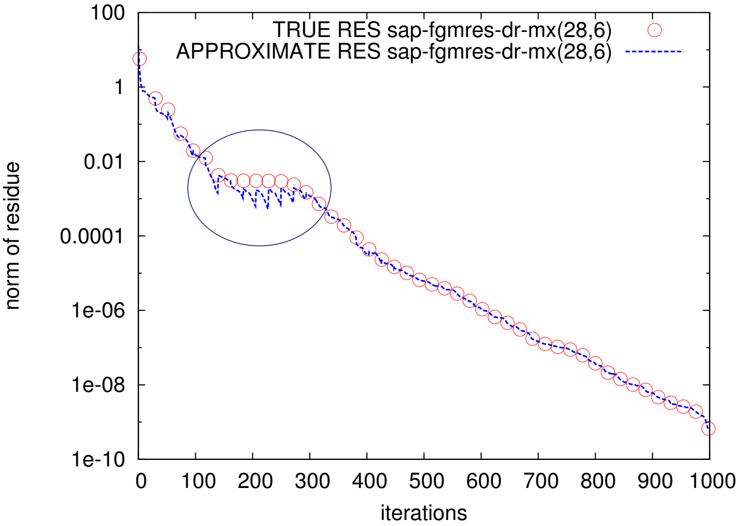
- Use information contained in the current Krilov subspace to approximate low eigenvectors
- Rebuild a new Krilov subspace with the approximate low eigenvectors
- Restart, improve current eigenvectors, and evetually find new ones
- Keep high convergence rate with a limited storage requirement

THE ALGORITHM AT WORK



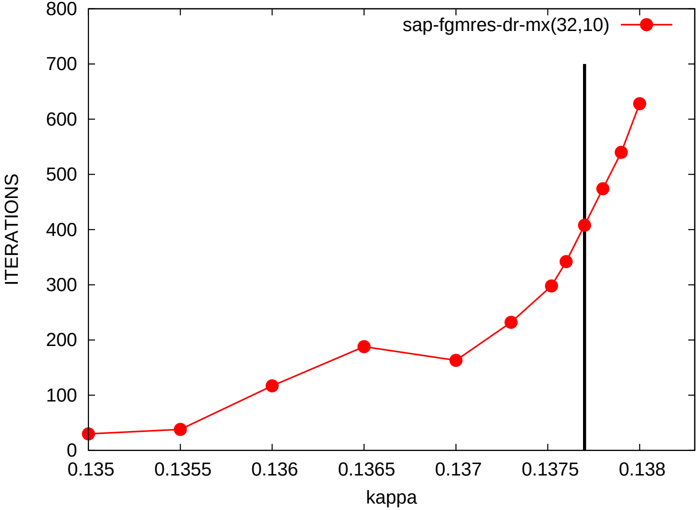
$16^3 \times 32$, $\beta = 5.29$, $\kappa_{sea} = 0.13500$, $\kappa_{val} = 0.13768$, $\kappa_{val}^c \approx 0.13770$

Eigenvectors development



$16^3 \times 32$, $\beta = 5.29$, $\kappa_{sea} = 0.13500$, $\kappa_{val} = 0.13768$, $\kappa_{val}^c \approx 0.13770$

Kappa dependence



Conclusions

Machine and code are working

- SAP preconditioner proved to be a good choice for QPACE
- We have a production-quality solver which is used on QPACE
- On a typical run with typical solver parameters the solver sustains 20% of the single precision peak
- SAP-FGMRES-DR is very promising with low quark masses

- Special thanks to: D. Pleiter, H. Simma, A. Frommer, Y.Nakamura, T. Streuer, S. Solbrig, B. Mendl, and the whole QPACE team!