

SciDAC Software Infrastructure for Lattice Gauge Theory

Jefferson Laboratory --- February 1, 2002

Software Co-ordinating Committee

- Rich Brower --- Boston University
- Carleton DeTar --- University of Utah
- Robert Edwards --- Jefferson Laboratory
- Don Holmgren --- Fermi National Laboratory
- Bob Mawhinney --- Columbia University/BNL
- Celso Mendes --- University of Illinois
- Chip Watson --- Jefferson Laboratory

SciDAC Software Infrastructure Goals

- *Create a unified programming environment that will enable the US lattice community to achieve very high efficiency on diverse multi-terascale hardware*

Major Software Tasks

- I. QCD API and Code Library
- II. Optimize Network Communication
- III. Optimize Lattice QCD Kernels
- IV. Application Porting and Optimization
- V. Data Management and Documentation
- VI. Execution Environment

Participants in Software Development Project

Bob Mawhinney	Columbia	----	
Chulwoo Jung	BNL	100%	Sept 1, 2001
Chris Miller	BNL	100%	
New hire	BNL	100%	(March 1,2002)
Don Holmgren	FNAL	40%	
Ron Rechenmacher	FNAL	25%	
Jim Simone	FNAL	35%	
Simon Epsteyn	FNAL	10%	
New hire	FNAL	100%	(Feb 1, 2002)
Steve Gottlieb	FNAL	10%	Oct 15, 2001
Daniel A. Reed	Illinois	----	
Celso L. Mendes	Illinois	35%	Oct 1, 2001

Robert Edwards	Jlab	----	
Chip Watson	Jlab	33%	
Chris McClendon	Jlab	100%	(Summer 01)
Jie Chen	Jlab	100%	Jan 1, 2002
Patrick Dreher	MIT	----	
Andrew Pochinsky	MIT	100%	
Richard Brower	BU	30%	
Massimo Di Pierro	BU	100%	(Oct 1, 2002)
Carleton DeTar	Utah	----	
James Osborn	Utah	50%	Sept 1, 2001
Doug Toussaint	Arizona	----	
Eric Gregory	Arizona	50%	Oct 15, 2001

Lattice QCD – extremely uniform

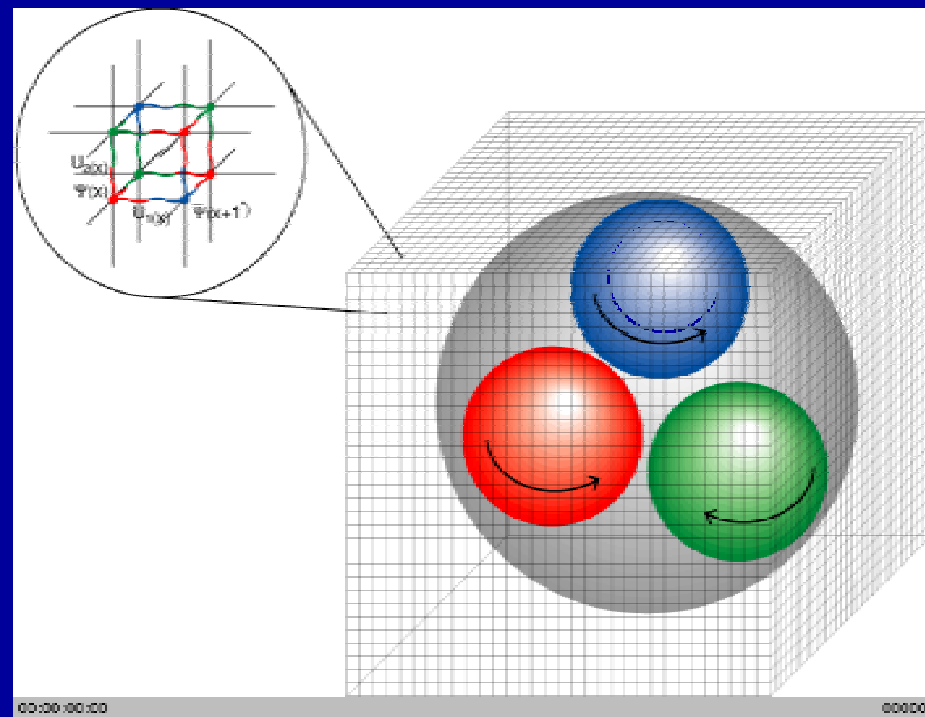
Dirac operator

$$D\psi(x) = \sum_{\mu} (\partial_{\mu} + igA_{\mu}(x))\psi(x)$$

Discretize on lattice – finite difference

$$D\psi(x) = \frac{1}{2a} \sum_{\mu} [U(x)\psi(x+\hat{\mu}) - U^{\dagger}(x-\hat{\mu})\psi(x-\hat{\mu})]$$

- Periodic or very simple boundary conditions
- SPMD: Identical sublattices per processor



QCD-API Level Structure

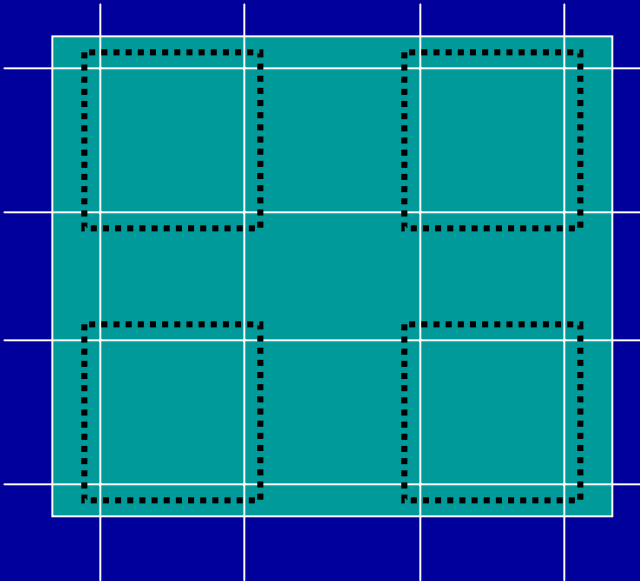
Level 3	
Dirac Operators, CG Routines etc. C, C++, etc. (Organized by MILC or SZIN or CPS etc.)	
QDP_XXX Level 2	
Data Parallel QCD Lattice Operations (overlapping Algebra and Messaging) A = SHIFT(B, mu) * C; Global sums, etc	
Lattice Wide Linear Algebra (No Communication) e.g. A = B * C	Lattice Wide Data Movement (Pure Communication, non-blocking) e.g Atemp = SHIFT(A, mu)
QLA_XXX	QMP_XXX
Level 1	
Single Site Linear Algebra API SU(3), gamma algebra etc.	Message Passing API (Know about mapping of Lattice onto Network Geometry)

I. Design & Documentation of QCD-API

- **Major Focus** of Software Co-ordinating Committee
 - See agenda and minutes on <http://physics.bu.edu/~brower/SciDAC>
 - Published documents to appear on <http://www.lqcd.org>
- **First design workshop**, Jlab: Nov. 8-9, 2001
- **Second design workshop**, Jlab: Feb 2, noon-midnight, 2002
 - finalize level 1 API and resolve outstanding questions for level 2 data parallel framework .
- **Goal:**
 - Distribute documentation and a C implementation for community review by Lattice 2002 in Boston, MA.
 - Foster “Linux style” contributions to level 3 API library functions?

Data-Parallel Programming Model

Data layout over processors



- Basic uniform operations across lattice: $C(x) = A(x) * B(x)$
- Map problem grid onto machine grid.
- API should hide subgrid layout and subgrid faces communicated between nodes.
- Like Fortran 90 – place Data Parallel paradigm on top of message passing.
- Implement API without writing a compiler.

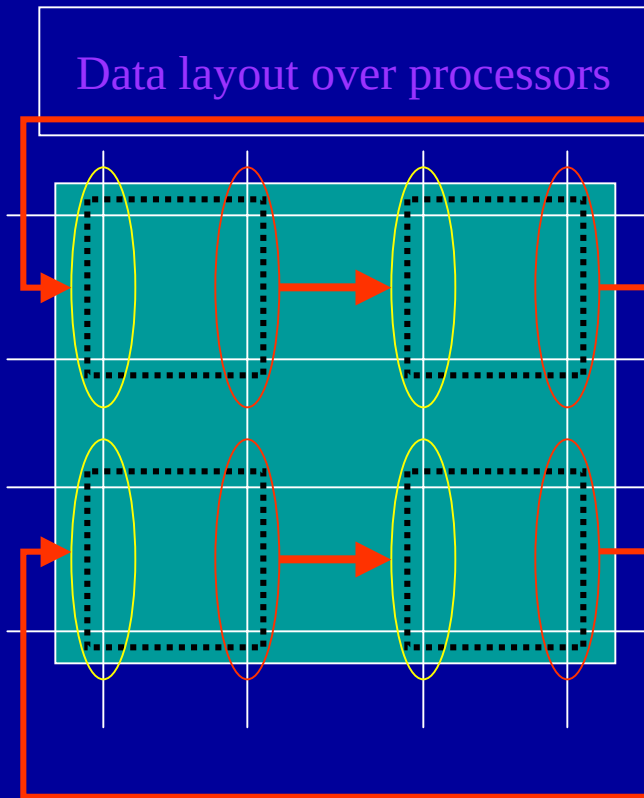
API Design Criteria

- Routines are **extern C functions** callable from C and Fortran: **extern functions** $\Leftrightarrow$ **C++ methods**.
- **Overlapping** of computation and communications.
- **Hide data layout**: Constructor, destructors. Query routines to support limited number of deftypes.
- **Support for multi-process** or multi-threaded computations hidden from user control.
- **Functions** do not (by default) make conversions of arguments from one layout into another layout. An error is generated if arguments are incompatible.

II. Level 1 MP-API implementation

- Definition of MP interface (Edwards, Watson)
 - Bindings for C, C++ and eventually Fortran.
 - see doc <http://www.jlab.org/~watson/lqcd/MessageAPI.html>
- Implementation of MP-API over MPI subset (Edwards)
- Implementation of C++ MP-API for QCDOC (Jung)
- Myrinet optimization using GM (Jie Chen)
- Port of MILC code to level 1 MP-API (DeTar, Osborn)
 - Status 80% complete pending finalization on Feb 2

Performance Considerations for Level 2



- Predefined channels are opened between neighboring nodes of a d dimensional grid.
- Overlapping communications and computations, e.g. $C(x) = A(x) * \text{shift}(B, \mu)$:
 - The face of a subgrid is sent non-blocking to a neighboring node, e.g. in the forward direction.
 - The neighboring node, in the backward direction, sends its face into a preallocated buffer.
 - While this is going on, the specified operation is performed on the interior sites.
 - A “wait” is issued and the operation is performed on the face.

Lazy Evaluation for Overlapping Comm/Comp

Consider the equation $\text{dest}(x) = \text{src1}(x) * \text{src2}(x + \text{nu})$; (for all x)

or decomposed as

$$\begin{aligned}\text{tmp}(x) &= \text{src2}(x + \text{mu}); \\ \text{dest}(x) &= \text{src1}(x) * \text{tmp}(x)\end{aligned}$$

Implementation 1: This can be implemented as two functions as follows:

```
Shift(tmp, src2, mu, plus);  
Multiply(dest, src1, tmp);
```

Implementation 2: If the Shift is changed to also return its result, this can be rewritten as:

```
Multiply(dest, src1, Shift(src2, mu, plus));
```

Communications-Computations Overlap

$T = \text{shift}(A, \mu)$ *sending face in μ & receive in $-\mu$*
 $C = T * B$ *calls wait on send/receive*

```
Type *QDPF_shift_T(Type *a, int mu)
{
    for(I=0; I < size_of_face; ++I)
        QDPF_copy_T1T2_repl(msgmem_f[mu][I], a[offset[I]])
    QMP_start(msgh_a[mu])      // start receive
    QMP_start(msgh_f[mu])      // start send
}

void QCDF_mult_T3T1T2_op3(Type3 *c, const Type1 *t,
                          const Type2 *b)
{
    thread_dispatch(&interior, interior_volume, c, t, b)
    QMP_wait(msgh_t)           // Wait on receive
    thread_dispatch(&face, face_size, c, t, b)
    QMP_wait(msgh_f)           // Wait on send
}

void interior(int start, int end, va_list args) // similar for face
{
    for(I=start; I < end; ++I)
        QDPF_mult_T3T1T2_op3_S(c[offset3[I]], t[offset1[I]], b[offset2[I]])
}
```

Data Types

- Fields have various types (indices):

Color: $U^{ij}(x)$, Spin: $\Gamma_{\alpha\beta}$, $\psi_{\alpha}^i(x)$, $Q_{\alpha\beta}^{ij}(x)$

- Index type (i.e the “fiber” over “base” lattice site)

- Gauge : $\text{Product}(\text{Matrix}(\text{Nc}), \text{Scalar})$
- Dirac: $\text{Product}(\text{Vector}(\text{Nc}), \text{Vector}(\text{Ns}))$
- Scalars: Scalar
- Propagators: $\text{Product}(\text{Matrix}(\text{Nc}), \text{Matrix}(\text{Ns}))?$

- Support Red/Black sublattices & other subsets (Mask ?)

- Support compatible operations on types:

$$U^{ij}(x) * \Gamma_{\alpha\beta} * \psi_{\alpha}^i(x) \longrightarrow \text{Matrix}(\text{color}) * \text{Matrix}(\text{spin}) * \text{Vector}(\text{color}, \text{spin})$$

C Interface for Level 2

- void QCDF_mult_**T3T1T2_op3**(Type3 *r, const Type1 *a, const Type2 *b)

- **T3, T1, T2** are short for the type **Type1, Type2** and **Type3** :

LatticeGaugeF,	LatticeDiracFermionF,
LatticeHalfFermionF,	LatticePropagatorF

- **op3** are options like

nnr	$r = a * b$	nnn	$r = -a * b$
ncr	$r = a * \text{conj}(b)$	ncn	$r = -a * \text{conj}(b)$
cnr	$r = \text{conj}(a) * b$	cnn	$r = -\text{conj}(a) * b$
ccr	$r = \text{conj}(a) * \text{conj}(b)$	ccn	$r = -\text{conj}(a) * \text{conj}(b)$
nna	$r = r + a * b$	nns	$r = r - a * b$
nca	$r = r + a * \text{conj}(b)$	ncs	$r = r - a * \text{conj}(b)$
cna	$r = r + \text{conj}(a) * b$	cna	$r = r - \text{conj}(a) * b$
cca	$r = r + \text{conj}(a) * \text{conj}(b)$	ccs	$r = r - \text{conj}(a) * \text{conj}(b)$

C++ Interface for Level 2

Unary operations: *operate on one source into a target*

```
Lattice_Field Shift(Lattice_Field source, enum sign, int direction);  
void Copy(Lattice_Field dest, Lattice_Field source, enum option);  
void Trace(double dest, Lattice_Field source, enum option);
```

Binary operations: *operate on two sources into a target*

```
void Multiply(Lattice_Field dest, Lattice_Field src1, Lattice_Field src2, enum option);  
void Compare(Lattice_Bool dest, Lattice_Field src1, Lattice_Field src2, enum compare_func);
```

Broadcasts: *broadcast throughout lattice*

```
void Fill(Lattice_Field dest, float val);
```

Reductions: *reduce through the lattice*

```
void Sum(double dest, Lattice_Field source);
```

III. Linear Algebra: QCD Kernels

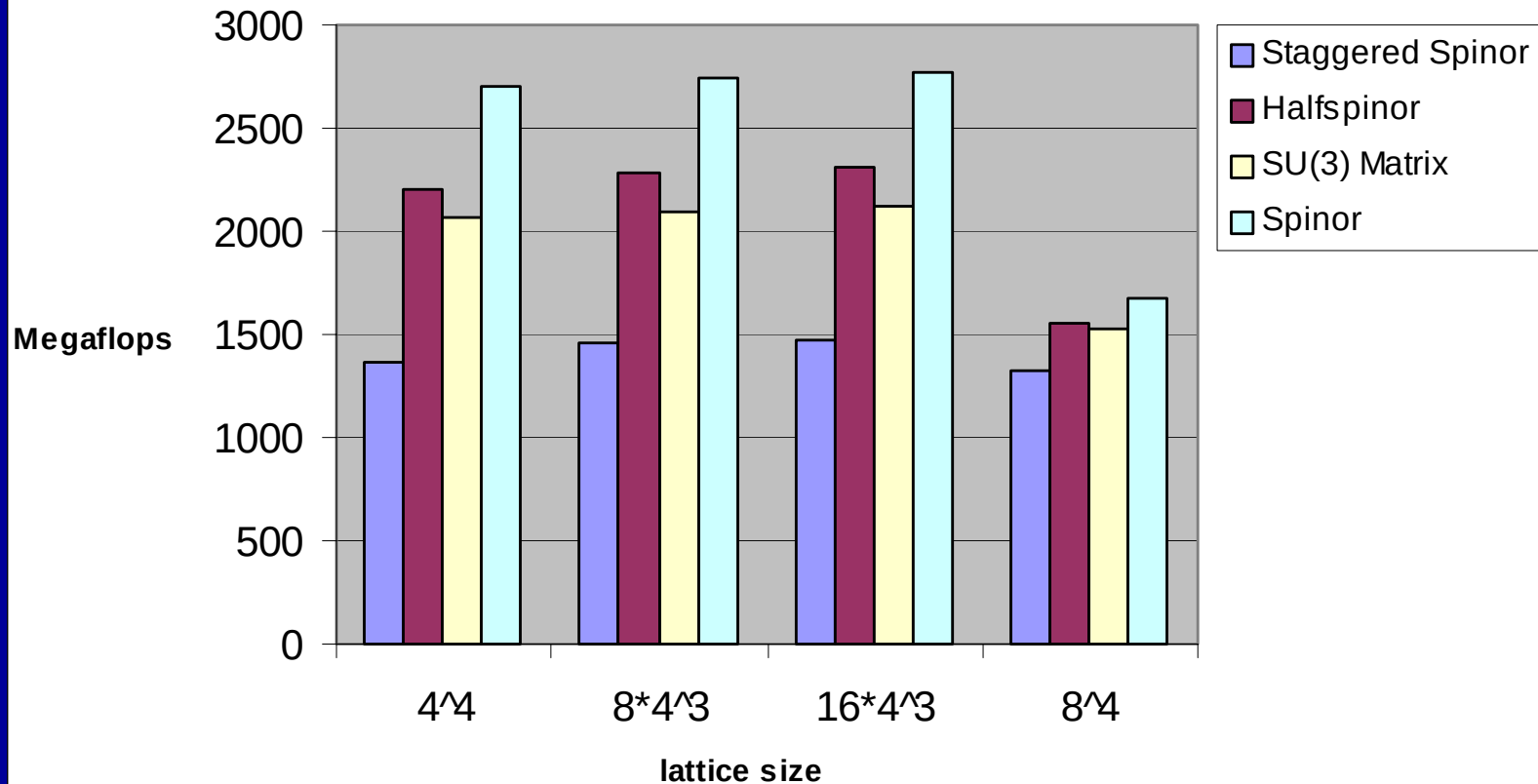
- **First draft of Level 1 Linear Algebra API** (DeTar, Edwards, McClendon)
 - http://www.jlab.org/~edwards/qcdapi/LinAlg1API_0_1.htm
- **Optimize on Pentium 4 SSE & SSE2 code:**
 - for MILC (Holgrem, Simone, Gottlieb)
 - for SZIN 1 (Edwards, McClendon)
 -
- **Vertical slice for QCD API** (Pochinsky)
 - API conformant example of Dirac CG
 - MILC implementation (Osborn)

Pentium I V with Streaming SIMD Extensions (SSE)

- **Streaming SIMD** (Single Instruction Multiple Data) Extensions which are length 4 vector instructions.
- **In Pentium III**, SSE has eight 128 bit registers used in single prec., so each vector register holds and operates on four 32 bit registers.
- **In Pentium IV**, SSE2 introduced which allows use of same registers in 64 bit (double prec.) mode.
- **Pentium IV** capable of four 32 bit operations per clock (e.g., two mult-adds) or two 64 bit operations (e.g., one mult-add) per clock.

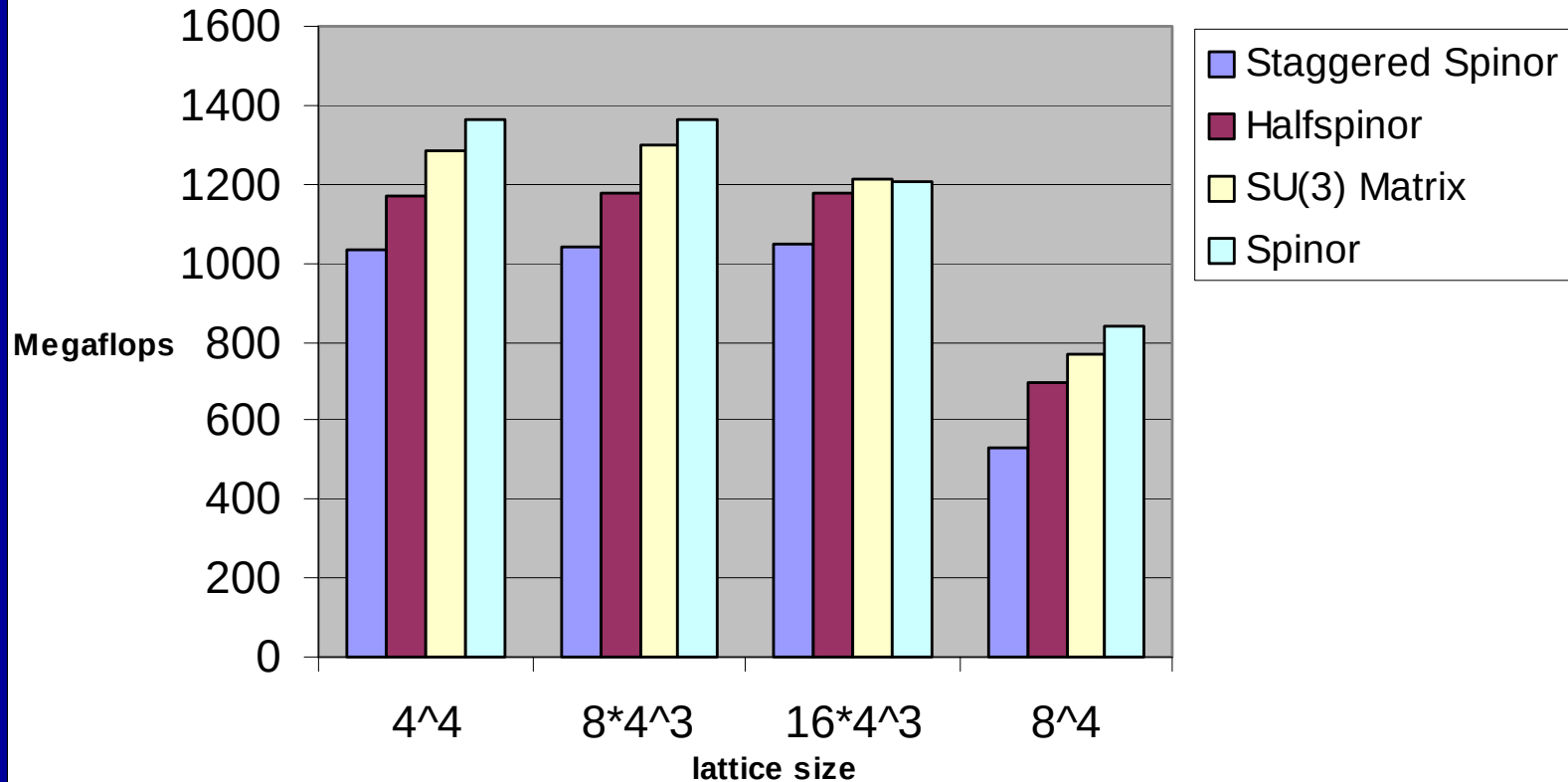
Level 1 Multiply Performance (32bit)

The Performance in Megaflops of 32-bit SU(3) Linear Algebra routines as a function of lattice size and operand size



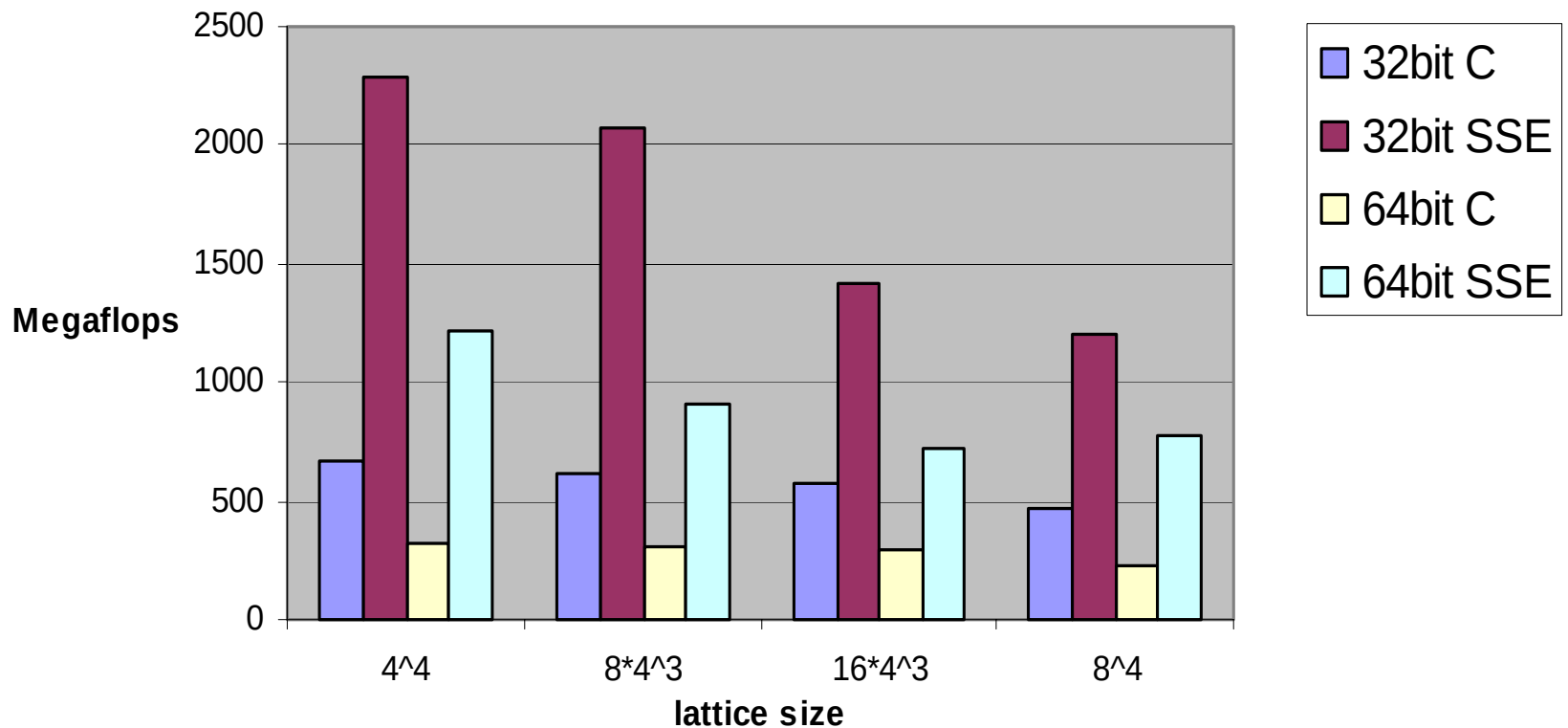
Level 1 Multiply Performance (64bit)

The Performance in Megaflops of 64-bit SU(3) Linear Algebra routines as a function of lattice size and operand size



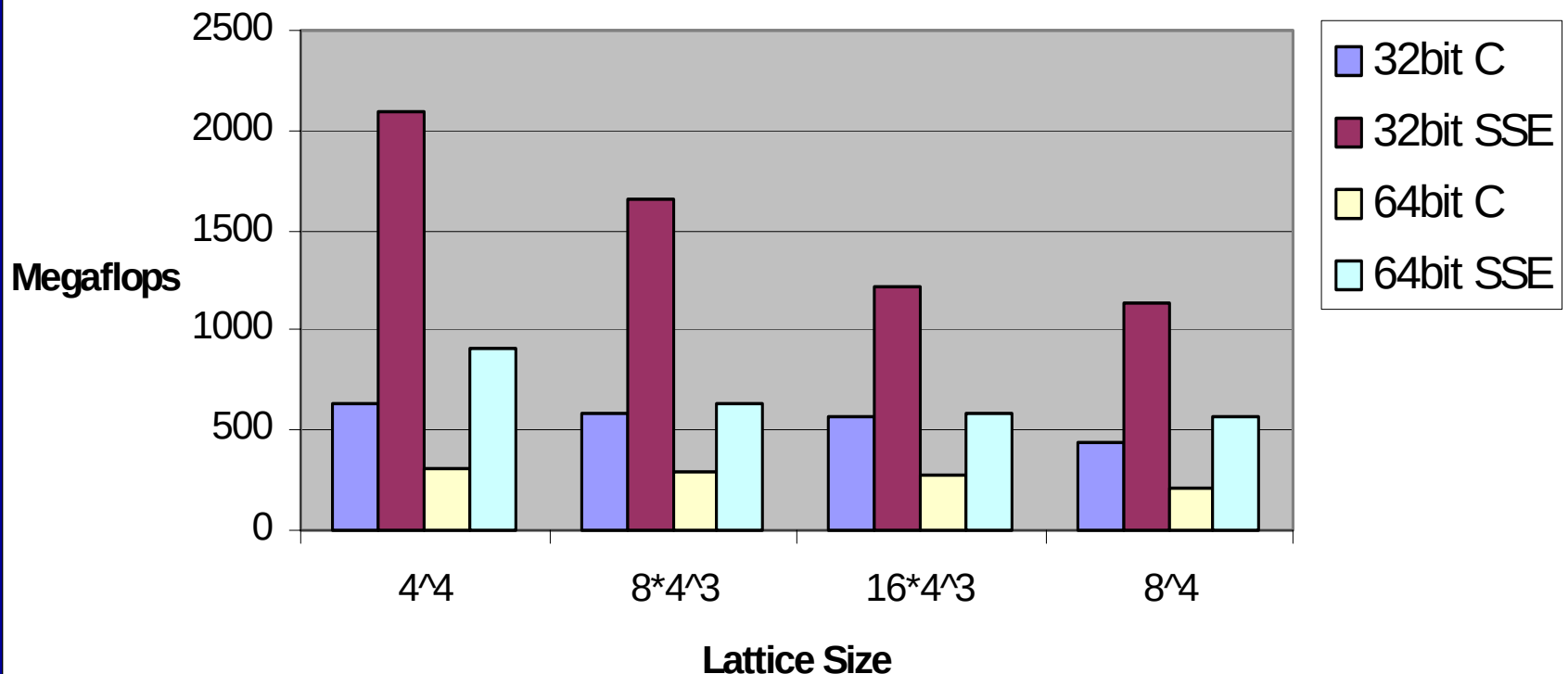
Single Node

Performance in Megaflops for single-node Dirac operator, 1.7 GHz Pentium 4 Xeon



Parallel Version on Single Node

Performance in Megaflops for Parallel Dirac Operator with no communications (upper bound performance per processor for cluster), 1.7 GHz Pentium 4 Xeon



IV. Application Porting & Opt.

- **MILC:** (revision version 6_15oct01)
 - QCDOC ASIC simulation of MILC (Calin, Christan, Toussaint, Gregory)
 - Prefetching Strategies (Holgren, Simone, Gottlieb)
- **SZIN:** (new documentation and revision (Edwards))
 - Implementation on top of level 1 API (Edwards, Pochinsky)
 - Status: efficient code for P4 by Spring 2002
- **CPS (Columbia Physics System)**
 - Software Testing environment running on QCDSP (Miller)
 - Native OS & fabric for MP-API (Jung)

V. Data Archives and Data Grid

- File formats and header
 - Build on successful example of NERSC QCD archive
 - Extend to include lattice sets, propagators, etc.
- Consider XML for ascii headers
 - Control input and output data files
 - Search user data using SQL to find locations.
- Lattice Portal
 - Replicate data (multi-site), global tree structure.
 - SQL-like data base for storing data and retrieving
- Web based computing
 - batch system and uniform scripting tool.

VI. New Performance and Coding Tools

- **Performance Analysis Tool:**
 - SvPABLO instrumentation of MILC (Celso)
 - Extension through PAPI interface to P4 architecture (Dongarra)
- **FNAL Tools:**
 - Trace Tools extension to Pentium 4 and instrumentation of MILC (Rechenmacher, Holmgren, Matsumura)
 - FNAL “rgang” (parallel command dispatcher)
 - FermiQCD (DiPierro)
- **Developing Q system** (Pochinsky)
 - High level code generator of optimized API library
 - Current version generates ANSI C code

SvPablo Instrumentation of MILC

The screenshot displays the SvPablo software interface, which is used for instrumenting and analyzing code. The main window is titled "svPablo" and contains several panels:

- Project Description:** MILC on IA-64 (version of April-2000)
- Source Files:** control.c, setup.c, update.c, g_congrad.c, com_mpl.c
- Performance Contexts:** IA-64 with 4 Processors, L=12
- Routines in Source File:** main, initialize_machine, g_sync, setup, setup_analyze
- Routines in Performance Context:** ks_congrad, dslash_spec, scalar_mult, mut_su3_m, mut_adj_su3_m
- Source File:** /u/ks/ks/pablo/Ceiko/MILC-INST/resources/control.c

The source code is displayed in a window, showing a loop for trajectory updates. The code includes comments and function calls, such as `g_source(EVEN);` and `m_iters=ks_congrad(F_OFFSET(phi), F_OFFSET(xxx), mass, niter, xsgain, EVEN, arsq);`.

Two "Specific Metric" dialog boxes are open, showing the following data:

- Specific Metric 1:** Call Statistics count: 6.0000 — update
- Specific Metric 2:** Call Statistics Duration: 710.888 — update

The "Legend: Source Code Metrics" panel on the right lists eight metrics with corresponding color-coded bars:

- Column 1: Call Statistics count (6)
- Column 2: Call Statistics Duration (710.888)
- Column 3: Loop Statistics count (1)
- Column 4: Loop Statistics Duration (848.401)
- Column 5: HW Statistics by Line Floating Point Instructions (7.12487e+09)
- Column 6: HW Statistics by Line Level 1 Data Cache Misses (2.8531e+09)
- Column 7: HW Statistics by Line Data TLB Misses (5.90391e+07)
- Column 8: HW Statistics by Line MFLOPS (134.706)

At the bottom of the interface, there are buttons for "Instrument/Clear Line" and "View Line Data".

QCD-API Status and Schedule

- **Level 1 Message Passing** API (MP-API) design completed
 - Implementation in MPI completed (Sept 2001)
 - C++ MP-API for QCDOC (Started)
 - Application port of MILC/SZIN to MP-API (Started)
 - Myrinet optimization in GM (Feb 2002)
- **Level 1 Linear Algebra** API 1st draft completed (Feb 2002)
- **Vertical Slice** prototype of level 2 (Feb 2002)
 - implementation for MILC (started)
- **Pentium IV** optimization of Linear Algebra (March 2002)