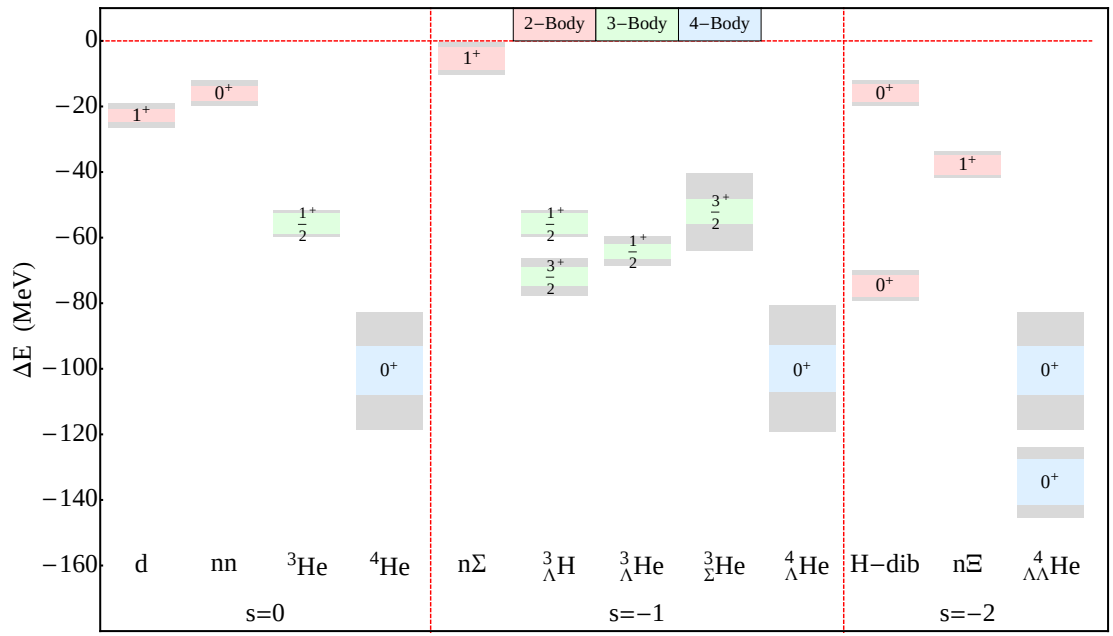
Emmanuel CHANG, University of Washington

SciDAC3 Postdoc

SciDAC
Scientific Discovery
through
Advanced Computing

INSTITUTE for
NUCLEAR THEORY

NPLQCD

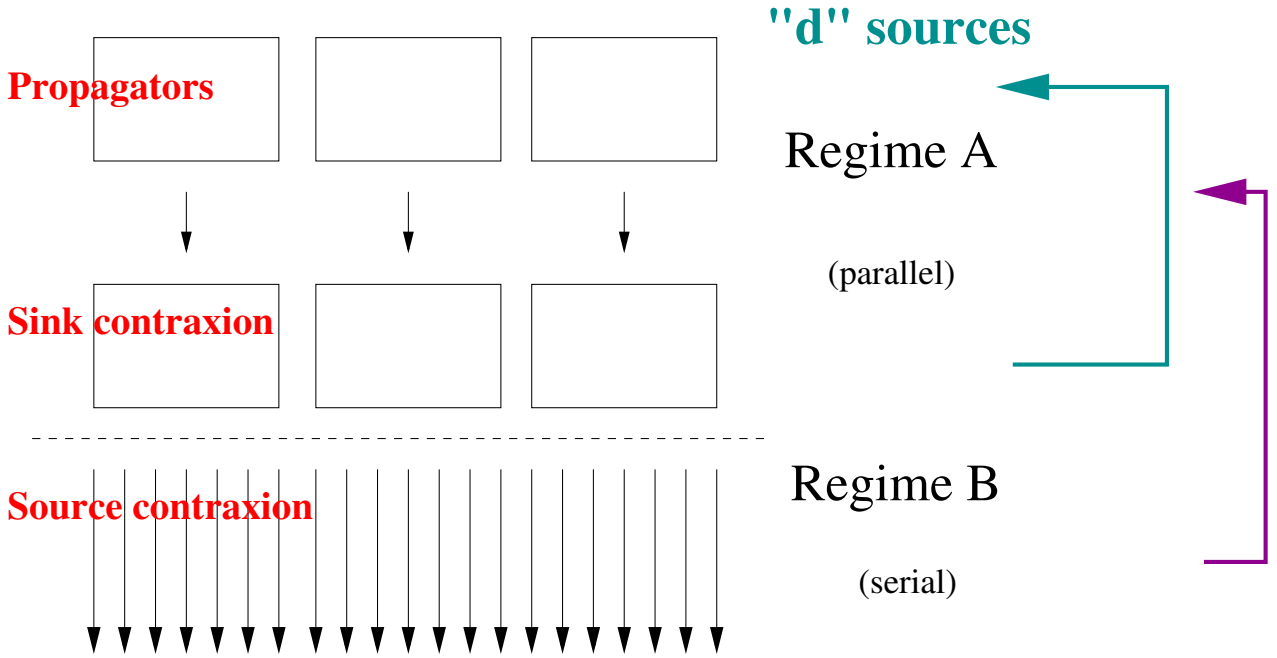


1. Disk space: too much or too little?
2. XML, SDB output files from Chroma.
3. Multigrid.
4. Alternative build procedure for USQCD modules.

Production: Running out of disk space?

Look for a simple solution to a scalability issue

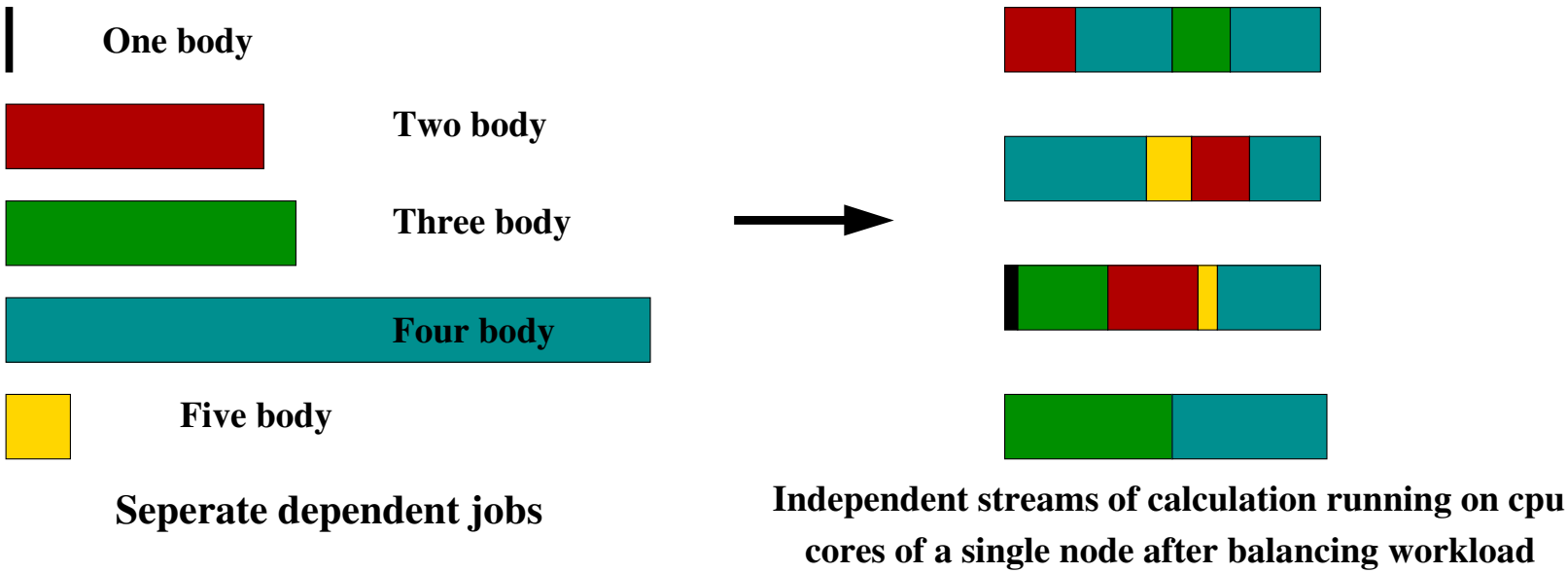
What does a typical job look like in our case?



Each job has groups of nodes working independently (*rectangles*) and cooperatively (*arrows*) at different stages and in different regimes.

Sink contractions produce *qqq blocks* as intermediate result, *stored on disk*, which are consumed by source contractions.

It is a little more complicated in practice. The arrows for the N-body contractions are of unequal length depending on the number of hadrons, number of momentum projections, number of distinct correlators of interest and complexity in the contractions.



In the simplest case, one implements the second stage (Regime B) by submitting dependent jobs from within stage one (Regime A).

For example, letting all cores on a node doing just 3-body contractions with each working on a different source.

Alternatively, each cpu core can work on the 1, 2, 3, 4 and 5-body contractions, in turn, for a given source. Suppose 16384 cpu cores are allocated for your job, you would expect to need at least **68 TB** of scratch disk space.

Performing 1, 2, 3, 4 and 5 body contractions in sequence on a single cpu core.

Using dependent jobs to maximize node usage doesn't solve the problem either.

If temporary disk space limited, simple solutions are to reduce wall clock time and/or reduce job size.

That is sort of like not managing to get on a metro in Paris during rush hours. You'll have to wait for the next train and hope that it is not full again.

A job does not get to run every time the scheduler checks what to run next. For a well managed job queue, on average it'll be fine but . . . .

Dependent jobs

With dependent jobs, there is an extra complication.

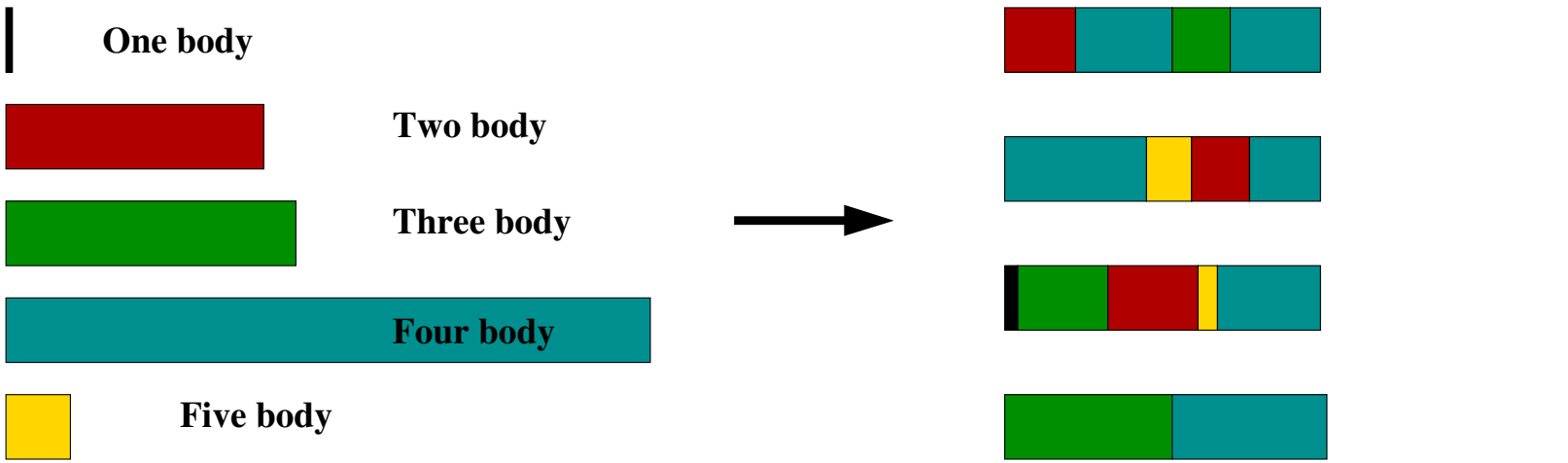
Typical job sizes:

Wall clock limit (hours)	Site	Cores per parallel task	simultaneous tasks per job
72	MareNostrum III	64	200
36	NERSC	384	50
24	J-Lab	128	16

At J-Lab, even with smaller job sizes, we permanently require over 100 TB of scratch space. Indeed, on a number of occasions those data get purged as pressure on that space increases, before the dependent jobs can start in a timely fashion to consume it.

This complicates the management of the production. Requesting for extra scratch space also requires returning computing time in exchange.

Key to the solution



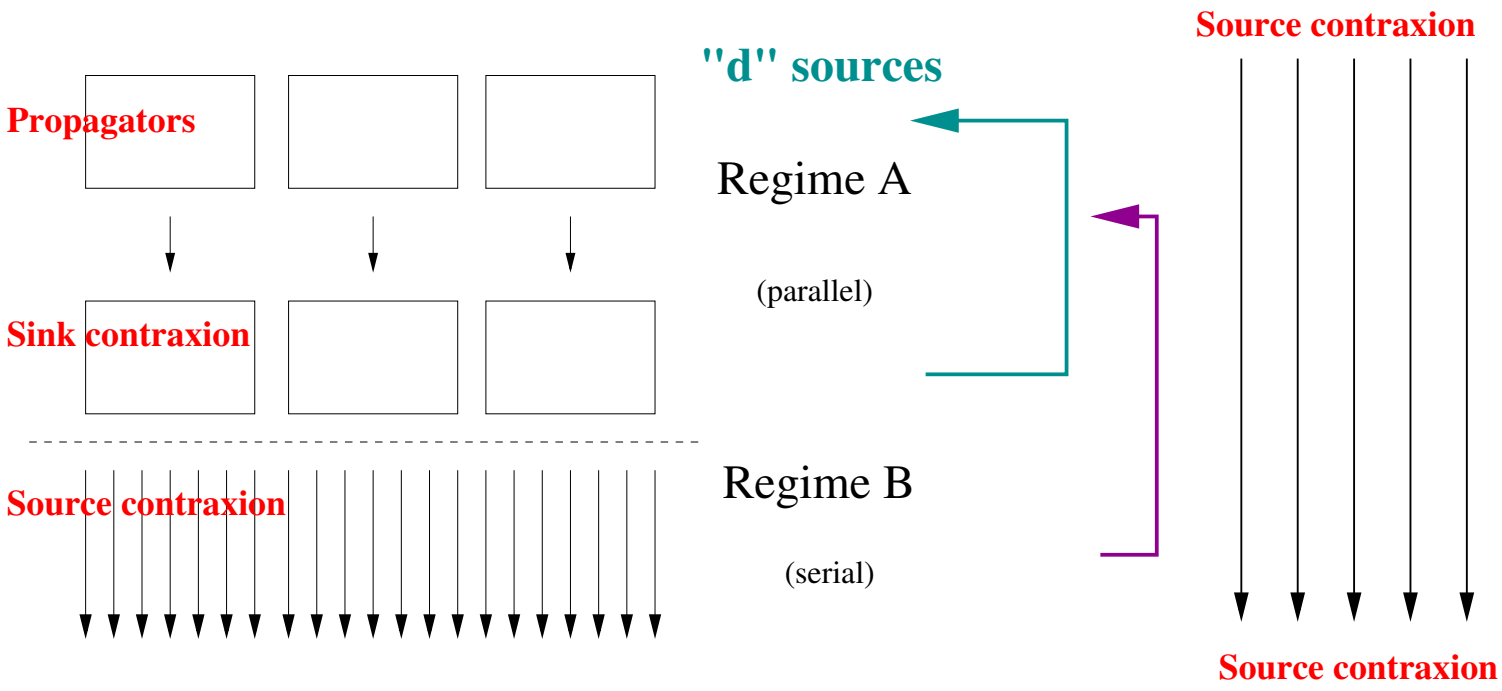
Seperate dependent jobs

Independent streams of calculation running on cpu cores of a single node after balancing workload

By shuffling and rescheduling all the N-body contractions with $\mathrm{N} = 1 \cdots 5$. We produce a regular pattern which we can fit into the workflow, presented on the first slide, consisting of one single job without dependents.

We no longer need to wait for multiple sources to complete the first stage of the calculation (qqq blocks) before we start the second stage.

With more coordination between the part A & B, you can further reduce the temporary disk space requirement by running streams of source contractions along side the original workflow.



As a interim solution, this is simple and effective. A more direct coupling of the two regimes without going through the disks is the obvious next step. The splitting at the boundary between sink/source contractions is also a choice and other possibilities exist.

Handling output:

Too big data or not too big data **or simply too many files?**

Once results from your simulations arrive on your laptop, the first thing is to check if any data is missing, list what is completed, plan the next run, ...

. . . which you can do by creating some tables

Check completed trajectories for a given correlator, e.g. `HyMag5`:

correlator	#src	qqq	stream	traj	B	parity	I	I3	J	Jz	S	irrep	MBS	p	p2cm	smear	source	sink
82318274d581293e0fd3e80f3a44d3bd	90	+	mn3-ec-0	2199	5	1	0	0	3/2	3/2	-3	H	HyMag5	0 0 0	0	P	7E441184	7E441184
614452fcdc8d293eefeaa0510a7ccb3d	92	-	mn3-ec-0	2199	5	1	0	0	3/2	3/2	-3	H	HyMag5	0 0 0	0	P	7E441184	7E441184
c6310a81882e2d3e0426b0dcf02ba8bd	90	+	mn3-ec-0	2205	5	1	0	0	3/2	3/2	-3	H	HyMag5	0 0 0	0	P	7E441184	7E441184
f92c23b6d2742e3e0923883797c9bc3d	94	-	mn3-ec-0	2205	5	1	0	0	3/2	3/2	-3	H	HyMag5	0 0 0	0	P	7E441184	7E441184
b17f4a977875283e9d5efde079cdce3d	94	+	mn3-ec-0	2210	5	1	0	0	3/2	3/2	-3	H	HyMag5	0 0 0	0	P	7E441184	7E441184
1c88b9b1ae2c283ed01f34f526e2c8bd	95	-	mn3-ec-0	2210	5	1	0	0	3/2	3/2	-3	H	HyMag5	0 0 0	0	P	7E441184	7E441184
fffcb3d042bb293ec000731ea2bf78bd	93	+	mn3-ec-0	2219	5	1	0	0	3/2	3/2	-3	H	HyMag5	0 0 0	0	P	7E441184	7E441184
bb7c69defde12a3e224da28b30e1a83d	96	-	mn3-ec-0	2219	5	1	0	0	3/2	3/2	-3	H	HyMag5	0 0 0	0	P	7E441184	7E441184

Check completed ensembles:

ensemble	stream	# distinct trajectory
cl21_32_96_b6p1_m0p2800_m0p2450	mn3-ec-0	384
cl21_32_96_b6p1_m0p2800_m0p2450	mn3-ec-1	384
cl21_32_96_b6p1_m0p2800_m0p2450	mn3-ec-2	384
cl21_32_96_b6p1_m0p2800_m0p2450	mn3-ec-3	384
cl21_32_96_b6p1_m0p2800_m0p2450	mn3-ec-4	192
cl21_32_96_b6p1_m0p2800_m0p2450	mn3-ec-5	192

For each table, you can formulate a question. You can ask the *oracle* questions about your data set and it will return the results in a table.

By giving constraints and conditions the answers should satisfy, the *oracle* will find them if they exist.

- Whatever form the data is in, you have to give a prescription to retrieve a specific piece of information.
- Usually the piece of information we seek, is stored as a consequetive bytes of data (1-2K bytes).

However the way in which these simple pieces of informations are held together can be quite arbitray; some facilitate access to these pieces of data, some obscure their presence by drowning them in noise.

Our data come in two forms:

format	category	classification	
XML	Hadron spectrum	TEXT	Mostly unstructured, closely mirror the way the code is structured
SDB	Multi baryon correlators	BINARY	J-Lab in house database

SDB stores key-value pairs, for example, key could be the concatenation of quantum numbers, momentum projection, ... of a particular correlator.

Multiple SDBs can be merged together directly. However individual correlators (1-2K) can also be extracted into seperate files.

Hadron spectrum data in XML have always been extracted into individual files ($\sim$ 400 correlators). These files basically work like key-value pairs. To get a particular correlator, one has to write out all of them to disk.

1. Each correlator $\sim$ 1 - 2K bytes.
2. A typical set of measurements has 200,000 - 500,000 sources.
3. Multiply and you get 80 - 200 millions of files easily for 4 - 10 TB of data.

To give an idea, what 100 million files are like:

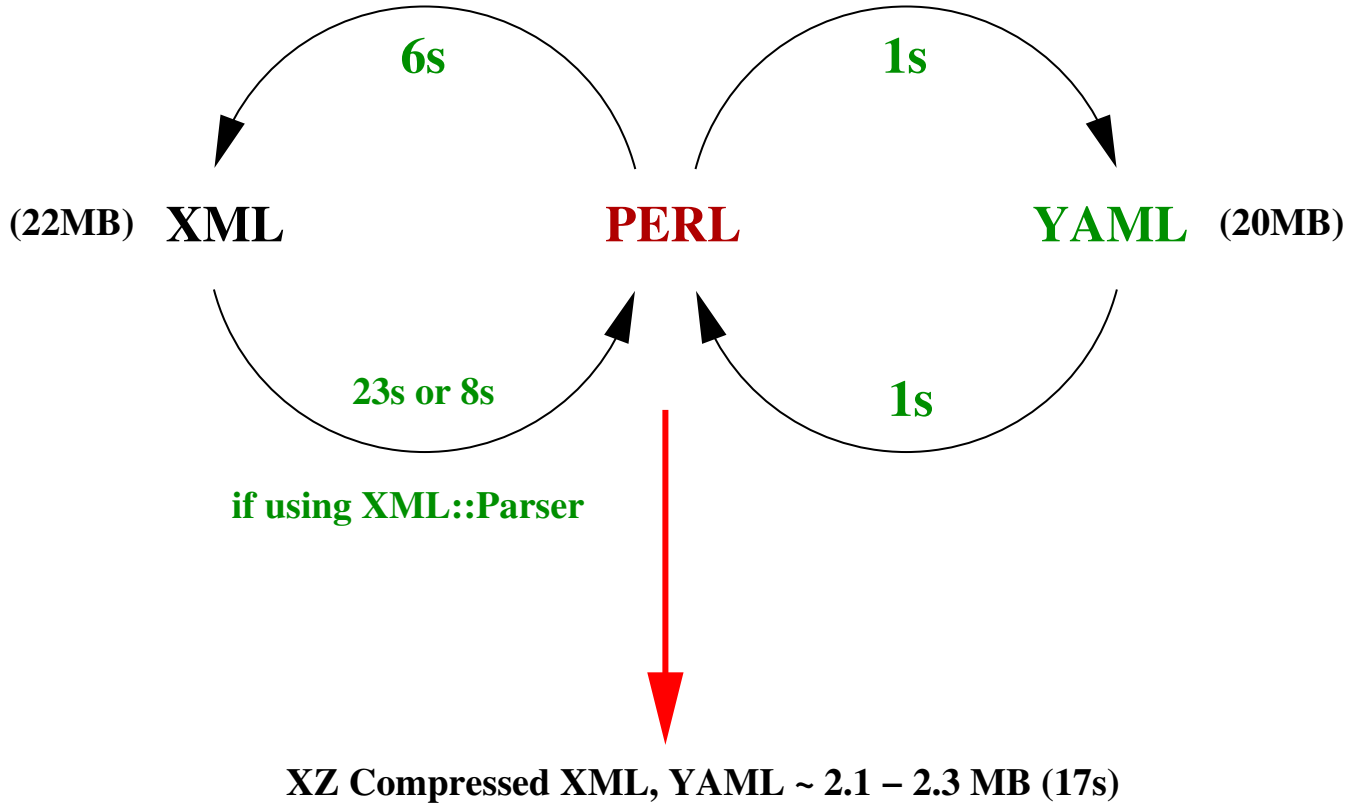
site	inode (millions)	disk space	
Hyak (UW)	210	130 TB	bkrs
MN3 (BCN)	268	1.6 PB	projects, scratch
J-Lab	878	1.16 PB	lustre
NERSC	500	3.54 PB	global scratch
My Ubuntu	56	770 GB	

Having to deal with large number of small files is a problem. For precisely this reason, the correlator data are put inside a big file. Kraken and NERSC lesson.

Organizing all these correlators inside a single file but without the tools to adequately describe them limits the usefulness of these files.

Key-value pairs are simply not enough. Besides, people do not necessarily follow the same convention for how the key should be constructed from say a given set of quantum numbers, ...

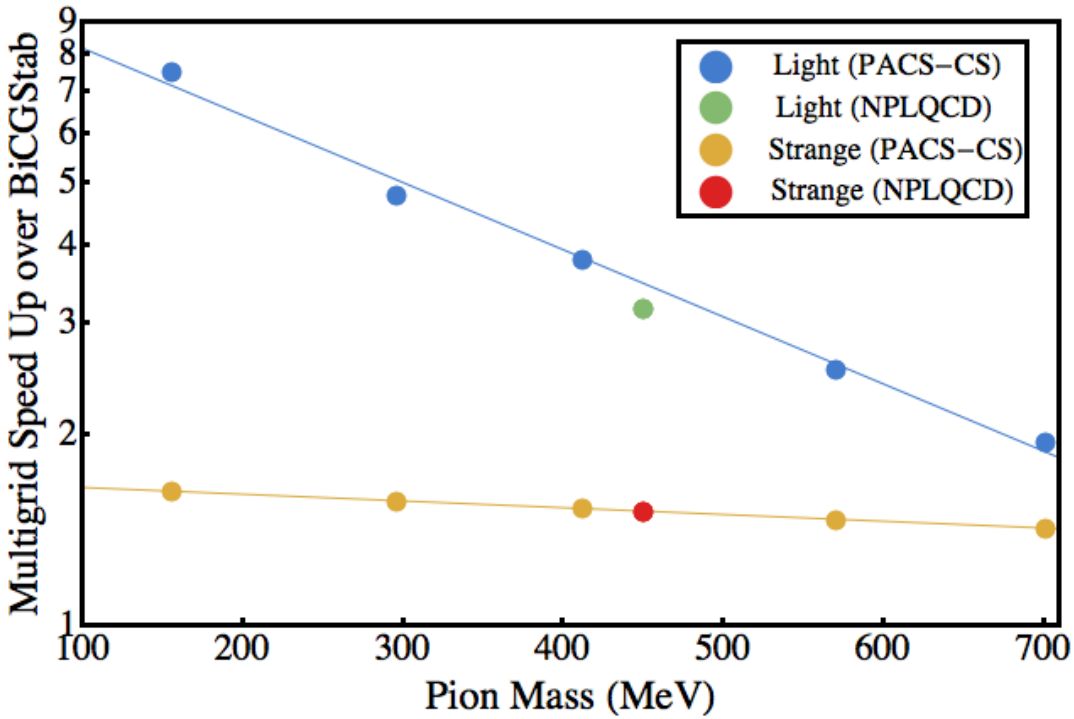
1. C/C++ program was written to convert SDB to SQLite3 databases using CppDB (http://cppcms.com/sql/cppdb/)
2. PERL was used to convert XML hadron spectrum files to SQLite3 databases.
3. A new *stripper*, in PERL, following exactly the convention used by the ADAT hadron spectrum stripper, was also written.
4. For current runs, the new recommended procedure is to immediately convert XML to YAML and compressed with XZ. The new stripper can read the compressed file directly and the saving in disk space is $\sim$ 90%.



Multigrid (MG)

1. New and cleaner Chroma interface to James Osborn's QDP/C and MG code (dated July 2012).
2. Cleaned up/refactored Saul D. Cohen's MG driver code (dated July 2012).
3. Added ability to create multiple multigrid from Chroma.
4. This is used in our $32^3 \times 96$ isotropic production on Mare Nostrum III in Barcelona.

I also compared MG with mixed precision BiCG-Stab for PACS-CS lattices at several light quark masses and one of NPLQCD's isotropic lattices at a pion mass of ~ 430 MeV.



As very limited user support was provided for multigrid, and the configure/Makefile for a complete Chroma integration was not yet available, it was necessary to

1. identify modules required to compile multigrid,
2. identify the dependecies between the required modules,
3. identify the correct preprocessor defines,
4. fixing some header files

In the process,

1. I understood why both `-geom` and `-qmp-geom` were necessary in order to set correctly the geometries for both Chroma and QDP/C+MG. This was necessary due to the different ways in which QDP/C and Chroma/QMP initialize and split the comunicators. This was fixed so that only `-qmp-geom` is needed. This makes more sense to me. The problem with inconsistent geometries in Chroma/QMP and QDP/C+MG initially caused some confusion and delays.
2. I optimized the compilation of QLA and QOP/C. Both generated functions with different combinations of precision and colours using PERL scripts. As the number of files generated is significant, with just a few functions per file, the large number of small files become a problem for Make to handle.

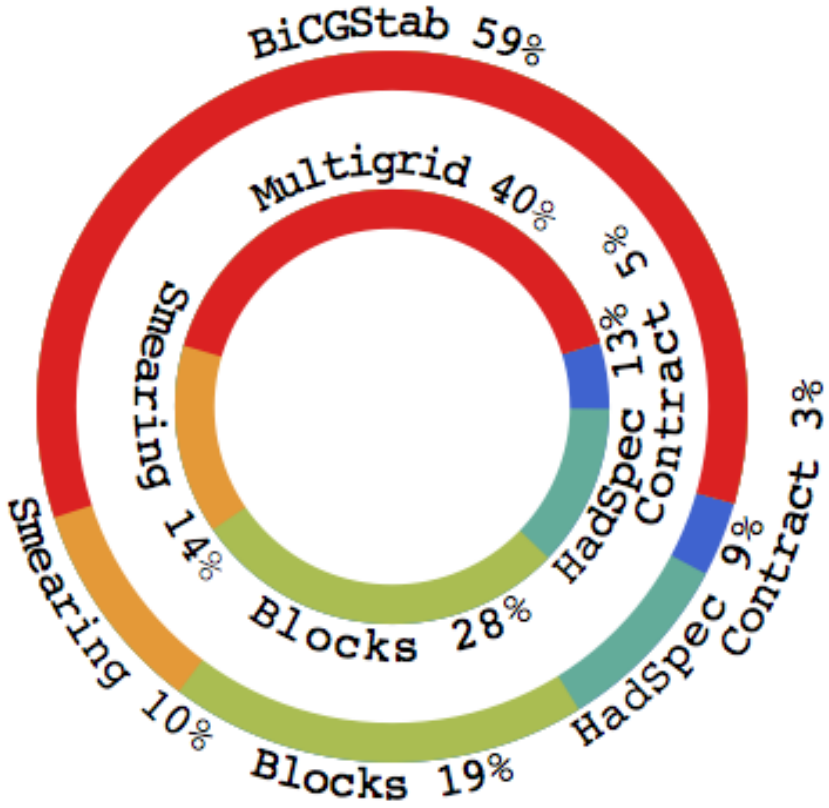
The way to proceed is very much like what I described in the parallelization of the source contractions. However here there's one difference. The preprocessor defines for different combinations of precision and colours are different. So you cannot mix them directly.

To get around this, you break each group into small enough chunks. You employ a single make file so that Make can schedule the compilation more efficiently. This is implemented using Makepp.

Finally, the use of a *single* make file is extended to all the modules required for building MG with Chroma. Again, implemented using Makepp.

I removed most of the *configure*s as much of that describes dependencies between modules inside the USQCD ecosystem and can be *calculated*. It is really just a matter of convention where you install a module.

Life isn't just about inverters.



PS:
Mathematica reads HDF5 that it writes. That's all.
Petition Wolfram to improve its HDF5 support.