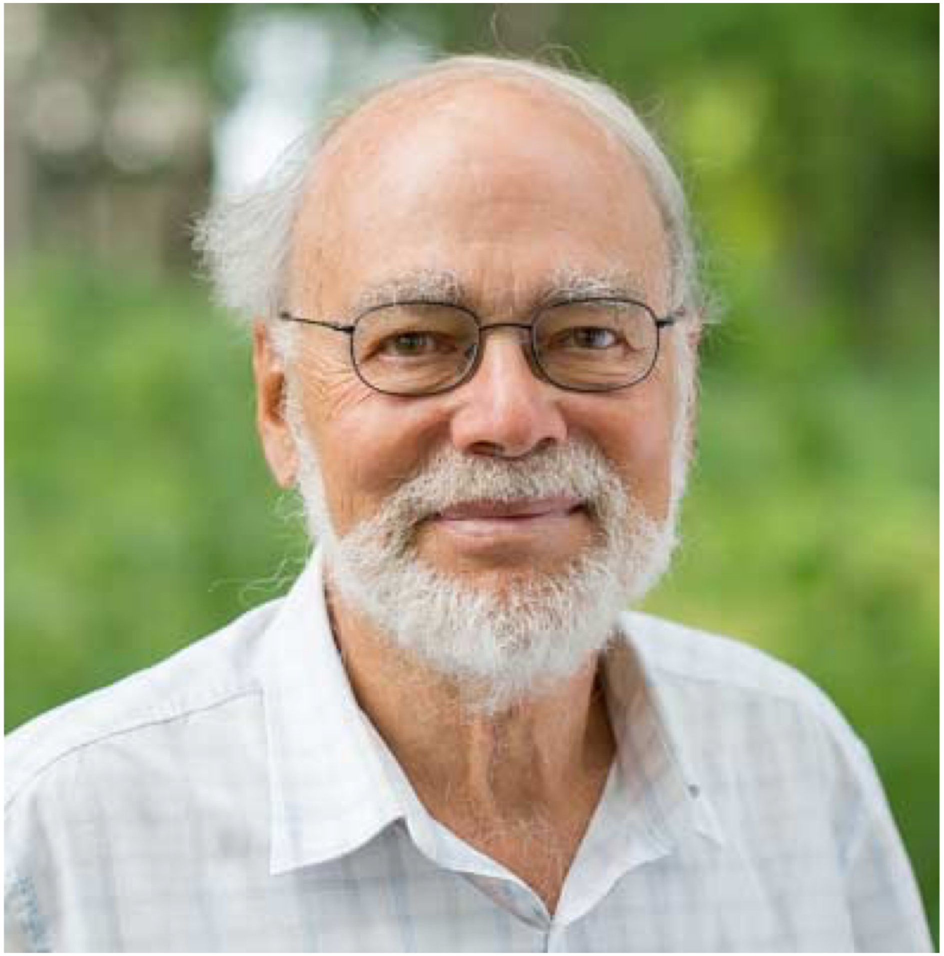
Computational Nuclear Physics
A Symposium in Honor of Steven C. Pieper



From Automated Theorem Proving to Nuclear Structure Analysis with Self-Scheduled Task Parallelism

(a personal history of one programming model)

Rusty Lusk (Ralph Butler)

Mathematics and Computer Science Division

Argonne National Laboratory

Theme

- *Application* programming models should be *simple*.
- Their *instantiations* might be more complex, and their actual *API's* might be even more so, not to mention their implementations.
- Also, the programming models of the *libraries* that implement them might be more complex.
- For example, the message passing *model* is simple: people are familiar with it from physical mail, phone calls, email, etc.
- There have been several *instantiations* (PVM, Express, EUI, p4, etc.) and multiple implementations of MPI.
  - As an application programming model, MPI is simple because applications use the simple parts
  - the more exotic parts of MPI are used by libraries to implement simple application programming models (or should be).
- MPI's full API is a really a system programming model, driven by library developers developing portable libraries that implement simple programming models for applications.

Example

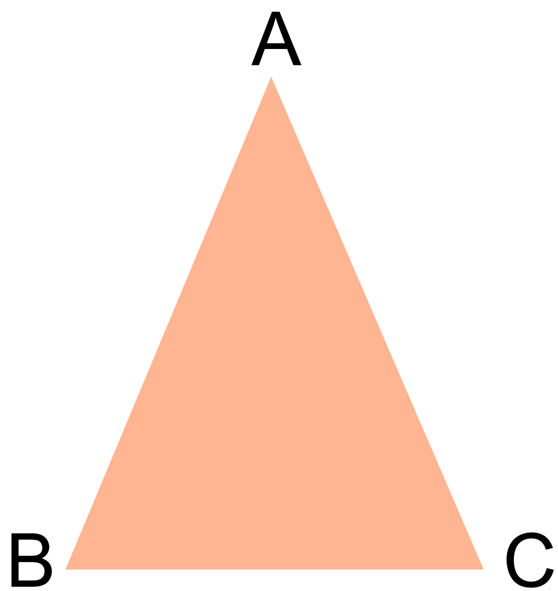
- Let’s discuss a simple programming model which has managed to remain simple through a number of instantiations and implementations.
- It is related to, but not the same as, several current task-based systems.
- It was how I wrote my first non-trivial parallel programs, back before the term “programming model” was in use (I didn’t know it was a programming model).
- I call it “self-scheduled task parallelism” (SSTP). My first work in computer science, after a stab at (very) pure mathematics, was in automated theorem-proving, at Argonne with Larry Wos, Ross Overbeek, and Bill McCune.
- The SSTP model was invented (not really on purpose) to parallelize the Argonne theorem prover (Otter).
- Therefore I am going to motivate it by entertaining you with a short introduction to automated theorem proving.

Outline

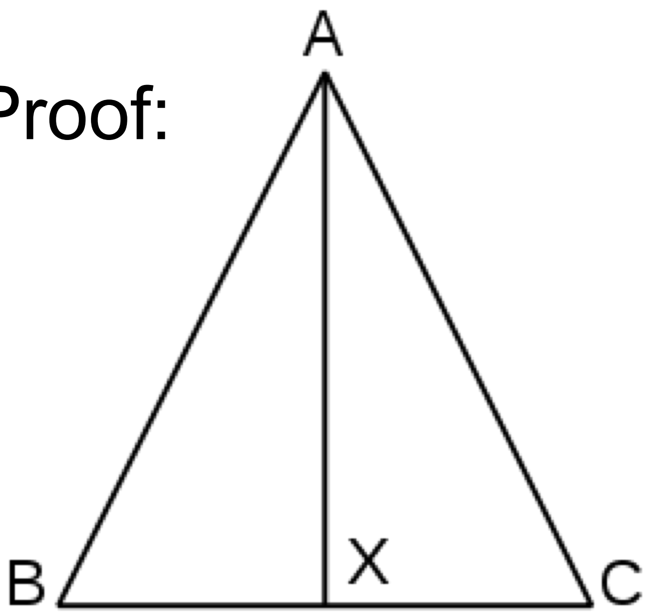
- Some ATP successes (why automated theorem proving is so much fun)
- Resolution-based automated theorem proving
  - How it works
  - A serial algorithm
  - Some parallel algorithms
- SSTP for a parallel Prolog system
- Why SSTP died out for a while
- Resurgence in Nuclear Physics SciDAC project as ADLB
  - Asynchronous Dynamic Load Balancing (ADLB), a minimal PM
  - ADLB is our current instantiation of the SSTP model
- Improving ADLB with another simple API, for memory management (DMEM)
- Recent results and current work

Going Way Back…

- Proposition 4 of Euclid's Elements (300 BCE) says that the base angles of an isosceles triangle are equal. This theorem is called the *Pons Asinorum**.



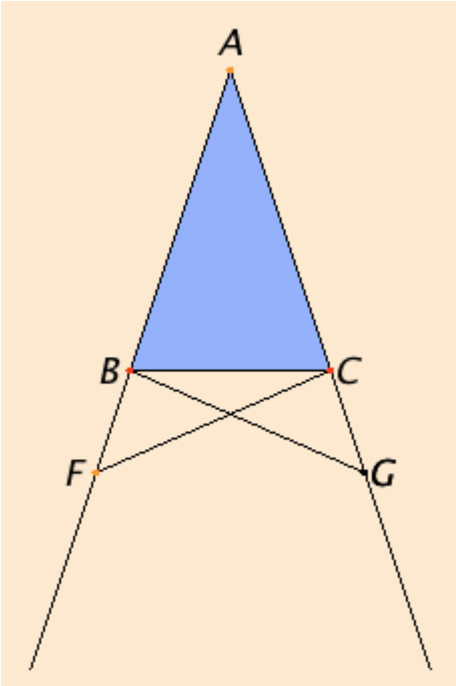
Proof:



Euclid

* "Bridge of Asses"

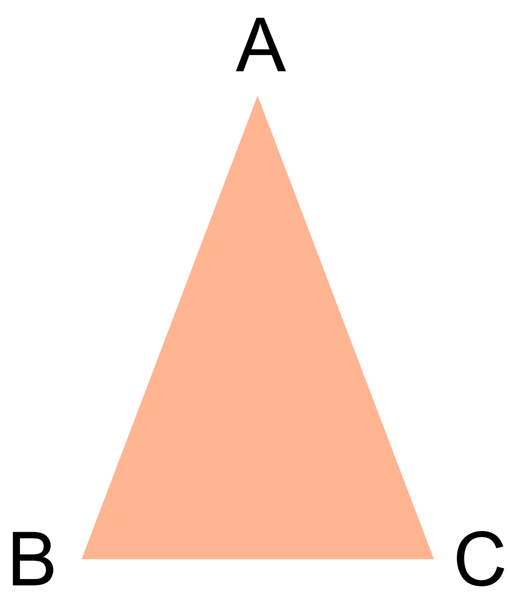
Euclid’s Proof of the *Pons Asinorum* (From the *Elements*)



- Since *AF* equals AG, and *AB* equals *AC,* therefore the two sides *FA* and *AC* equal the two sides *GA* and *AB,* respectively, and they contain a common angle, the angle *FAG.*
- Therefore the base *FC* equals the base *GB,* the triangle *AFC* equals the triangle *AGB,* and the remaining angles equal the remaining angles respectively, namely those opposite the equal sides, that is, the angle *ACF* equals the angle *ABG,* and the angle *AFC* equals the angle *AGB.*
- Since the whole *AF* equals the whole *AG,* and in these *AB* equals *AC,* therefore the remainder *BF* equals the remainder *CG.*
- But *FC* was also proved equal to *GB,* therefore the two sides *BF* and *FC* equal the two sides *CG* and *GB* respectively, and the angle *BFC* equals the angle *CGB,* while the base *BC* is common to them. Therefore the triangle *BFC* also equals the triangle *CGB,* and the remaining angles equal the remaining angles respectively, namely those opposite the equal sides. Therefore the angle *FBC* equals the angle *GCB,* and the angle *BCF* equals the angle *CBG.*
- Accordingly, since the whole angle *ABG* was proved equal to the angle *ACF,* and in these the angle *CBG* equals the angle *BCF,* the remaining angle *ABC* equals the remaining angle *ACB,* and they are at the base of the triangle *ABC.* But the angle *FBC* was also proved equal to the angle *GCB,* and they are under the base.

A better proof, found by an automated theorem proving program in the 70's

- Triangle BAC is congruent to triangle CAB by the side-angle-side theorem. Corresponding angles of congruent triangles are equal. QED.




PAPPI
ALEXANDRINI
MATHEMATICAE
Collectiones.
A FEDERICO
COMMANDINO
VRBINATE
In Latinum Conuersæ, & Commentarijs
Illustratæ.
VENETIIS.
Apud Franciscum de Franciscis Senensem.
M. D. LXXXIX.

- Also Pappus, 320 CE

Otter Proof (2019)

- ---------------- PROOF ----------------
- 1 [] -T(x,y,z) | T(x,z,y).
- 7 [] -T(x,y,z) | S(y,z).
- 14 [] -S(x,y) | SameLen(x,y,y,x).
- 22 [] -SameLen(x,y,z,u) | SameLen(u,z,y,x).
- 27 [] -Congruent(x,y,z,u,v,w) | SameAngle(x,y,z,u,v,w).
- 30 [] -T(x,y,z) | -T(u,v,w) | -SameLen(x,y,u,v) |
- -SameLen(y,z,v,w) | -SameLen(z,x,w,u) |
- Congruent(x,y,z,u,v,w).
- 32 [] -SameAngle(a,b,c,a,c,b).
- 34 [] T(a,b,c).
- 35 [] SameLen(a,b,a,c).
- 40 [hyper,34,7] S(b,c).
- 46 [hyper,34,1] T(a,c,b).
- 51 [hyper,40,14] SameLen(b,c,c,b).
- 56 [hyper,35,22] SameLen(c,a,b,a).
- 72 [hyper,56,30,34,46,35,51] Congruent(a,b,c,a,c,b).
- 85 [hyper,72,27] SameAngle(a,b,c,a,c,b).
- 86 [binary,85.1,32.1] $F.

clauses given	40
clauses generated	248
clauses kept	85
clauses fwd subsumed	198
clauses back subsumed	0
user CPU time	0.00
wall-clock time	0

A More Recent Example

- The following open question was posed to our group by Irving Kaplansky, big-cheese algebraist at the University of Chicago:
- Is there a finite semigroup that has an anti-automorphism but no involution?
- Our program proved not only was that answer was "yes," but that the smallest was of order 7 and there were four such.
- Getting results publishable in math journals was even more fun than doing college algebra homework problems and theorem-proving benchmarks.

Kaplansky

A More Recent Recent Example ☺

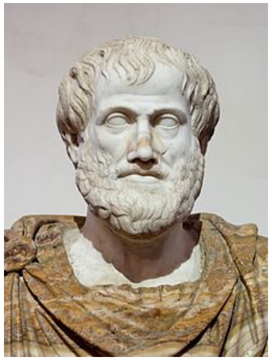
- In 2016, Google's DeepMath group did experiments on 32,524 theorems from the TPTP library of theorems.
- They shared their dataset with me, so I used a bit of Python3 code to convert their files to a format suitable as input to McCune's prover9.
- I was able to prove 31,498 of the 32,524 using prover9, giving each theorem a maximum of 90 seconds to find a proof. (Many of them run in under 1 second.)
- We hope to use the results of these experiments in Deep Learning projects related to ATP.

How Resolution Theorem Proving Works

- Convert statements in 1st-order logic into disjunctive normal form, in which all variables are universally quantified and disjunction is the only connective. Implications become disjunctions:
- $\forall x, P(x) \longrightarrow Q(x) \quad \Rightarrow \quad -P(x) \lor Q(x)$
- $\exists x, P(x) \quad \Rightarrow \quad P(a)$ (Skolem constant)
- Derive a new clause from 2 existing clauses by "cancellation":

$$\frac{\begin{array}{l}\cancel{-P}\ \text{v}\ Q\\ \cancel{P}\end{array}}{\qquad Q}$$

How It Works, continued



Aristotle

- Variables get instantiated to make the match:

All men are mortal.
Socrates is a man.
____
Socrates is mortal

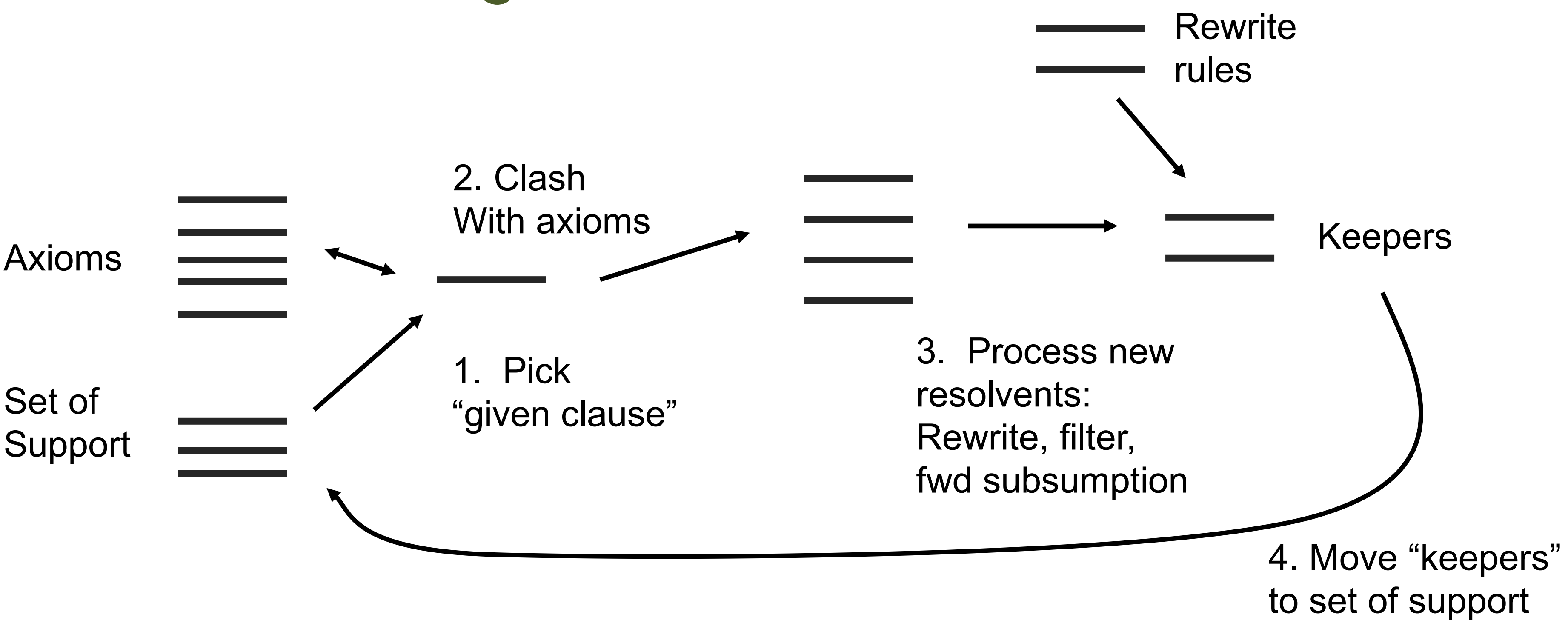
-Man(x) v Mortal(x)
Man(Socrates)
____
Mortal(Socrates)

- To prove a theorem, state its denial and derive a contradiction, denoted by the “null clause.”

P(a)
-P(a)
____
“ “

- The tricky bits are to avoid deducing too much and controlling redundancy

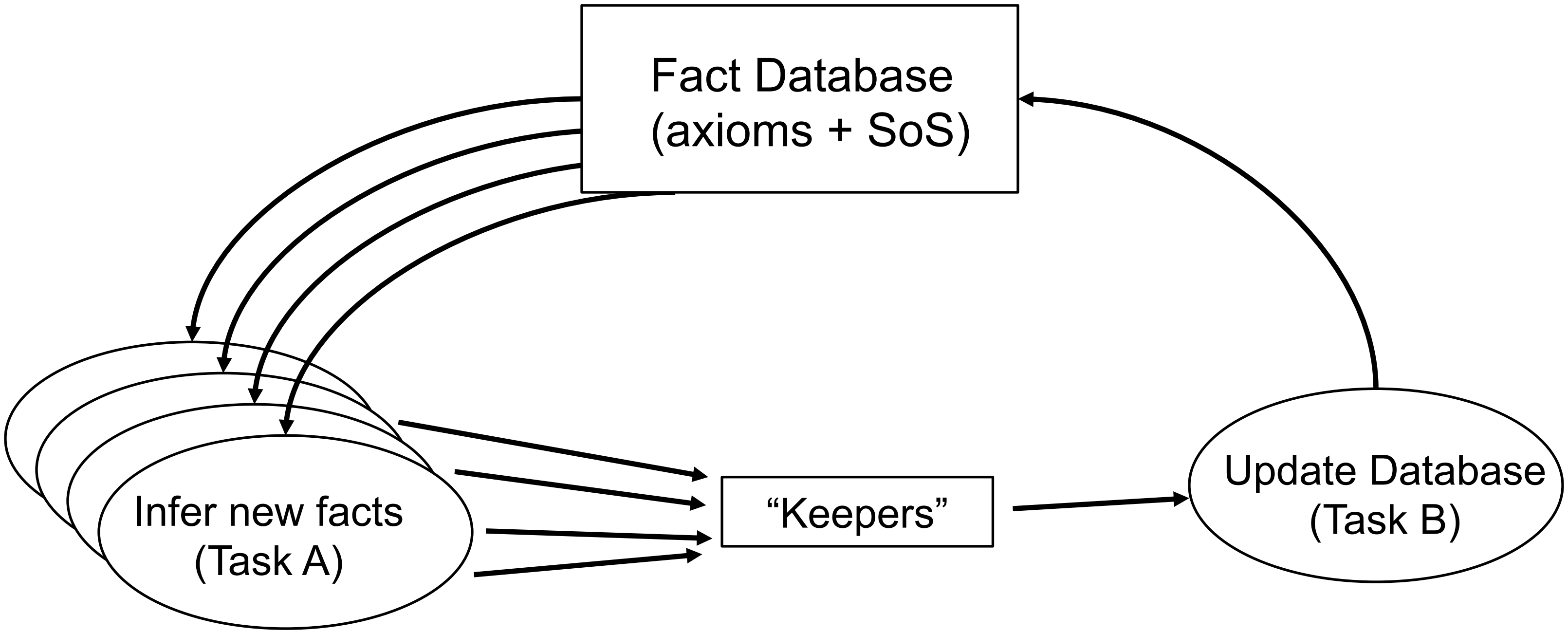
Otter's Basic Algorithm



Repeat until you deduce the empty clause, SoS becomes empty, or you run out of time or memory.

- A *very* irregular computation
- First attempt at parallelism: process new resolvents in parallel
  - No good, since not enough parallelism, barrier before each new given clause
- Next version, process multiple given clauses in parallel

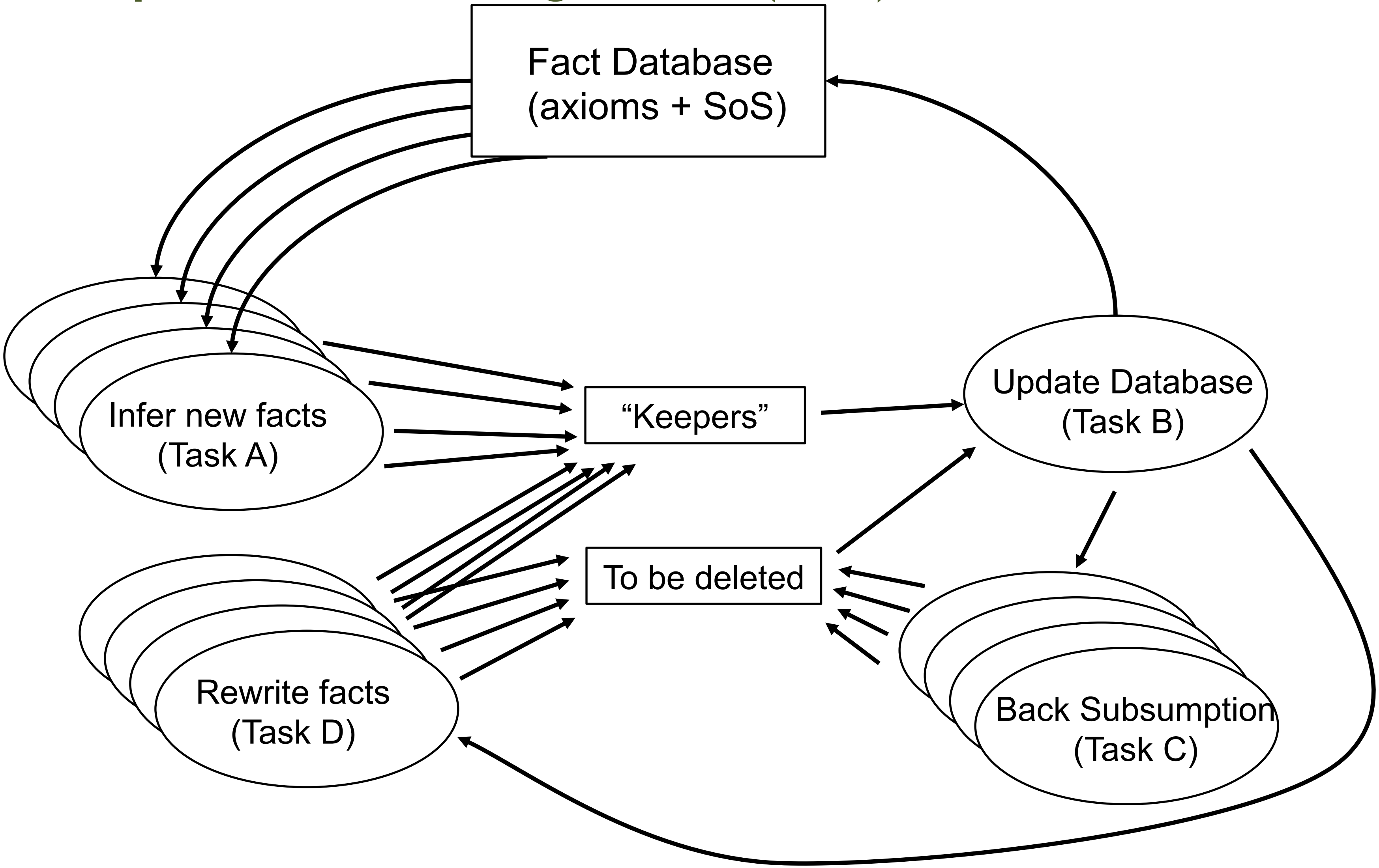
A Parallel Algorithm Without Deletion



- Task A: Pick a given clause and carry out steps 1-3 from previous slide
- Task B: For each clause in K, do final forward subsumption test and add to set of support
- All processes:
  - If Keepers list is non-empty and no other process is doing Task B, do Task B
  - Else do Task A

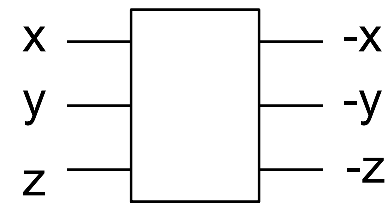
This is the origin of SSTP.

A Complete Parallel Algorithm (Roo)



- Only one process at a time does B, the rest is a free-for all with no traffic cop or DAG
- Again, each process executes same loop, acquiring work, doing it, making new work

Some Old (But Good) Results



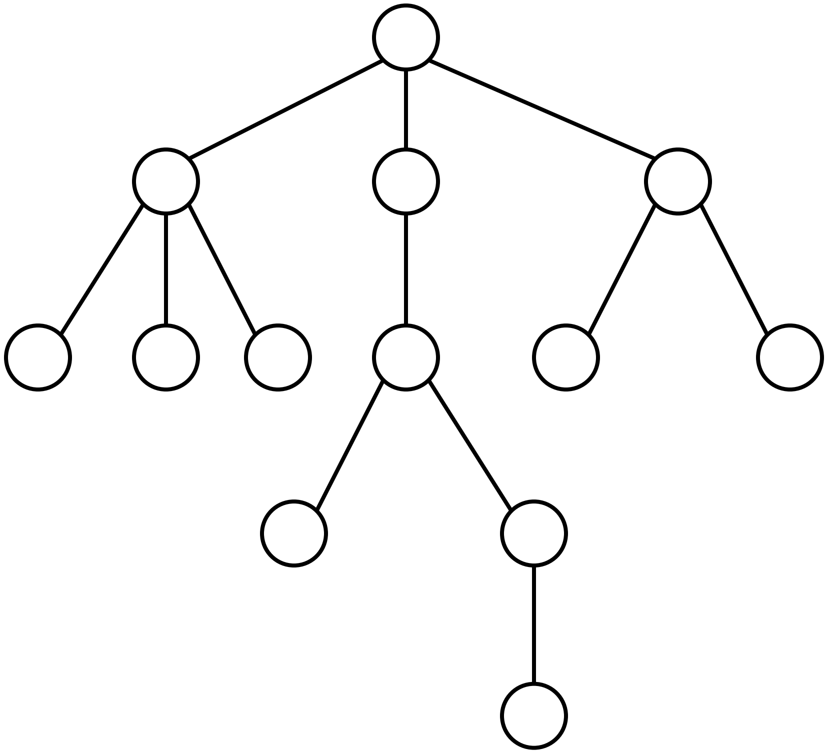
- The ”two inverter” problem:
  - Design a circuit, using AND, OR, and just two NOT gates, whose 3 outputs are the inversions of its three inputs.
- In implicational propositional calculus, the law of hypothetical syllogism can be derived from a proposed single axiom by condensed detachment:
  - -P(x) v -P(i(x,y)) v P(y) (Condensed detachment)
  - P(i(i(i(x,y),z), i(i(z,x), i(u,x) (Lukasiewicz axiom)
  - -P(i(i(a,b), i(i(b,c), i(a,c)))) (Denial of hypothetical syllogism)

	Otter-2inv	Roo24-2inv	Otter-Luka	Roo24-Luka
Runtime (sec.)	47236	2237	29098	1269
Generated	6323644	6351410	6706380	7108289
Kept	21342	21343	20410	18759
Speedup	1.0	21.1	1.0	22.9

- On 24-processor Sequent

Parallel Prolog

- Creating/acquiring work is again done by modifying a shared data structure
- Just beginning to identify and abstract these operations into general putting work into, and getting work out of, a shared work pool



SSTP Takes a Vacation

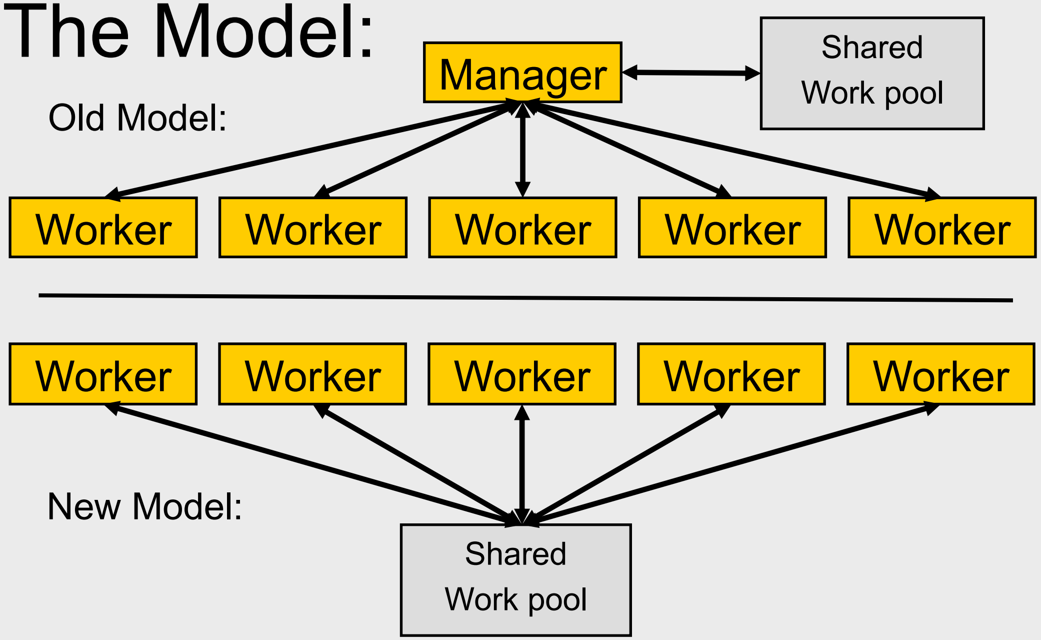
- As the number of processors multiplied, shared memory couldn't scale, and large-scale parallel computing went to message passing.
- DOE lost interest in inference as the hope of a program verification miracle faded.
- SSTP evolved (backwards) into the manager-worker programming model (e.g. Linda)
- This solved beautifully the load-balancing problem for irregular computations but hit its own scalability problem
  - Too many workers for a single manager to keep up with
  - Too little memory for a single manager to store the structures defining the work pool

Green's Function Monte Carlo – A Complex Application

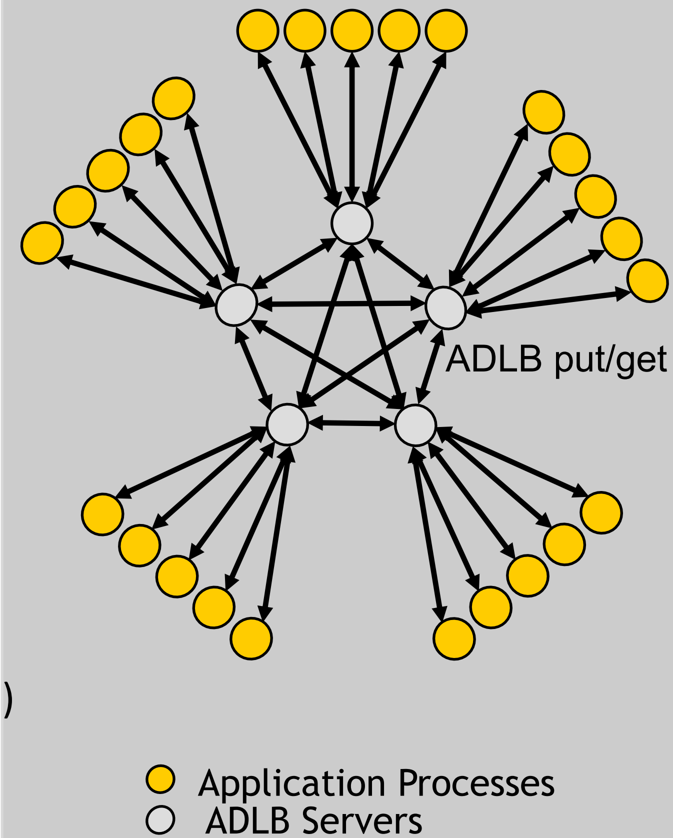
- Green's Function Monte Carlo -- the "gold standard" for *ab initio* calculations in nuclear physics at Argonne (Steve Pieper, Physics Division)
- A non-trivial manager/worker algorithm, with **assorted work types** and priorities; multiple processes create work dynamically; large work units
- Had scaled to 2000 processors on BG/L, then hit scalability wall.
- Needed to get to 10's of thousands of processors at least, in order to carry out calculations on ^{12}C, an explicit goal of the UNEDF SciDAC project.
- The algorithm threatened to become even more complex, with more types and dependencies among work units, together with smaller work units. An **extremely irregular computation**.
- Wanted to maintain original manager/worker structure of physics code
- This situation brought forth the Asynchronous Dynamic Load Balancing Library (**ADLB**), giving up generality for scalability and ease of use.
- Achieving scalability has been a multi-step process
  - balancing processing
  - balancing memory
  - balancing communication
- Now runs with 100's of thousands of processes

ADLB On One Slide

The Model:



An Implementation:



The **API**:

- ADLB_Put(type, priority, len, buf, target_rank, answer_dest)
- ADLB_Reserve(req_types, handle, len, type, prio, answer_dest)
- ADLB_Get_Reserved(handle, buffer)
- and a few housekeeping calls…

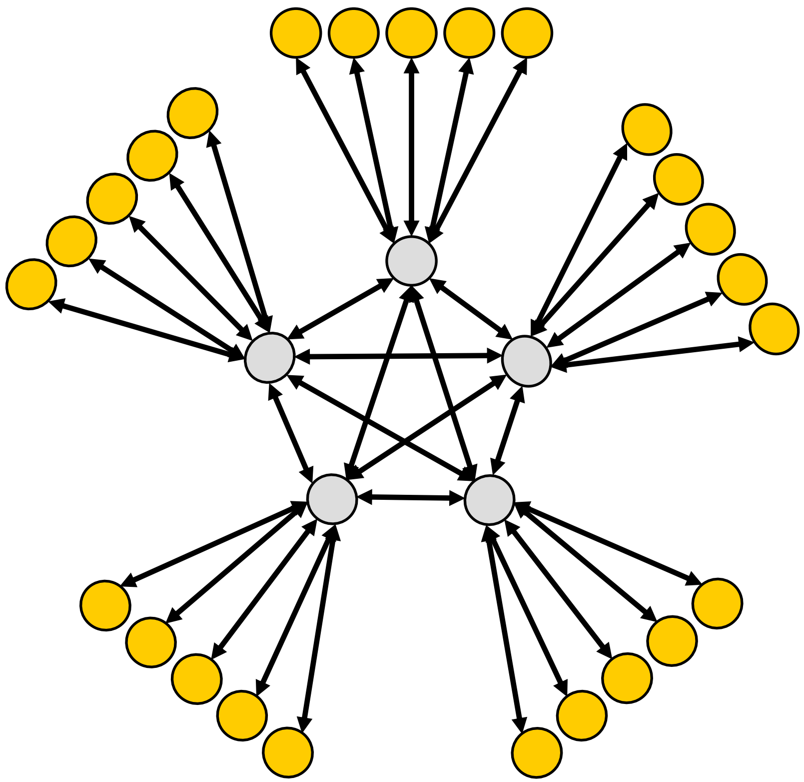
ADLB abstracts the idea of creating/acquiring work using put/get of work units into a work pool

ADLB Uses Multiple MPI Features

- ADLB_Init returns separate application communicator, so application processes can communicate with one another using MPI as well as by using ADLB features.
- Servers are in MPI_Iprobe loop for responsiveness.
- MPI_Datatypes for some complex, structured messages (status)
- Servers use nonblocking sends and receives, maintain queue of active MPI_Request objects.
- Queue is traversed and each request kicked with MPI_Test each time through loop; could use MPI_Testany. No MPI_Wait.
- Client side uses MPI_Ssend to implement ADLB_Put in order to conserve memory on servers, MPI_Send for other actions.
- Servers respond to requests with MPI_Rsend since MPI_Irecvs are known to be posted by clients before requests.
- MPI provides portability: laptop, Linux cluster, BG/Q, Cray
- MPI profiling library is used to understand application/ADLB behavior.

A Recent Problem and Its Solution

- The multiple servers were originally introduced to spread the communication (and computational) load that were swamping the one master.
- But they also store the data for the work units.
- As the work units became larger, we needed more servers for their storage capability, exacerbating the synchronization problem.
- Solution: decouple work unit allocation from work unit storage.



DMEM – A library to provide a shared-memory model on a distributed-memory machine

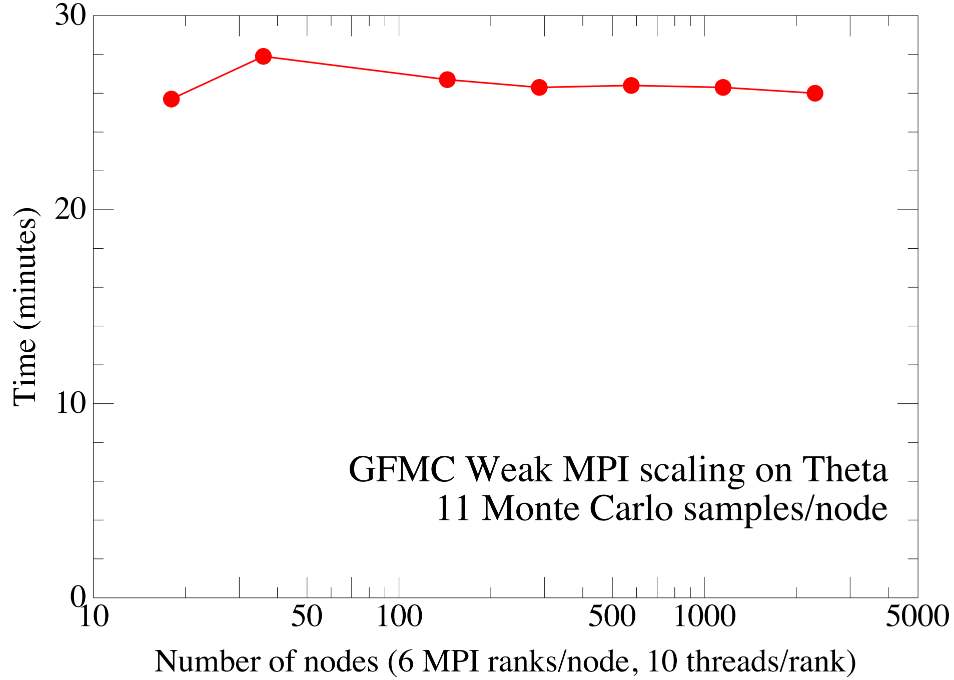
- **API** summary: put, get, copy, free, get-part, update
- User (application or another library) refers to a memory object via a (small) *handle*, which encodes its location and size.
- DMEM runs as a *separate thread* in applications, sharing memory with application processes, so local operations are fast.
- Optimization: put and copy operations are local if possible.
  - For non-local operations, multiple optimization strategies are possible
- Looking ahead, object size is of type MPI_Aint, which is typically a long int in C and an integer*8 in Fortran.

How DMEM Helps ADLB

- DMEM’s MPI communicator contains all of GFMC’s client (application) processes (not the servers).
- GFMC is modified to store work units containing DMEM handles instead of the large blocks of data that used to be the work units; data is stored and retrieved via DMEM_Put and DMEM_Get.
- All of application processes’ total memory is now available.
- Work units presented to ADLB are tiny (contain handles instead of entire work unit data).
- So way fewer servers are needed for storage.
- So ADLB’s synchronization challenge disappears.
- Everybody wins!

Sample Results for GFMC with DMEM on Theta

- Theta is new Knights Landing – based machine at Argonne, with 64 cores/node, and we have just started experiments
- GFMC is hybrid: OMP + ADLB + DMEM
  - Strong OMP scaling per node up to # cores (1 MPI rank on node)
  - Better throughput with multiple ranks per node
  - Weak scaling with ADLB up to current size of machine



Each rank uses 9 OMP threads and one pthread for DMEM; 6 ranks per node

A Lurking Future Problem (LFP)

- (Near) future machines are going to have lots of memory per node (for huge work units) and lots of threads (hardware and software) per node (to work on them).
- What if an ADLB (or even just a DMEM) application wants to utilize **work units whose size is larger than 2 GB** (approximately the size of a 32-bit integer)?
- ADLB and DMEM are agnostic about the internal structure of work units, so their internal messages use MPI_BYTE as their message type, so the count argument in MPI communications is the size (in bytes) of the message.
- MPI_{Send/Recv} specifies the count argument as an integer (still 32 bytes on most systems).
- The MPI-3 forum decided not to change this, because "long" messages could be sent/received on an MPI-compliant implementation by using MPI datatypes to lower the count argument into the 32-bit range.
- But:
  - Some people (even me, an MPI enthusiast) consider MPI datatypes inconvenient.
  - Some important MPI implementations are not MPI-compliant! (e.g. Mira and Titan)
- Solution: a long-message library for anyone who needs it: MPIL
  - Looks like MPI, except for MPIL_Count in MPI_Send/Recv, etc.
  - Limited version (enough for DMEM) working now

Summary

- Automated theorem proving, an irregular computation, motivated our initial self-scheduling, load-balancing approach
- ADLB, its current instantiation, demonstrates that by giving up some generality, a programming model can provide scalability without complexity for (some) applications.
- GFMC motivated ADLB, which motivated DMEM, which motivated MPIL.
  - But all 3 are small, portable, independent libraries
- DMEM was a big help to ADLB, but is potentially useful in a more general context. (e.g. to exploit multiple types of memory in a hierarchical memory system). Needs wider user input.
- MPIL will be a simple, portable way to provide long message support to any MPI program at lowest cost.
- Even little-bitty libraries (i.e. with small API's) can be useful in HPC physics applications (as long as they have a Fortran interface, of course).
- Automated theorem proving might be currently somewhat out of fashion, but wouldn't it be great if we could....

The End

Make America Logical Again!