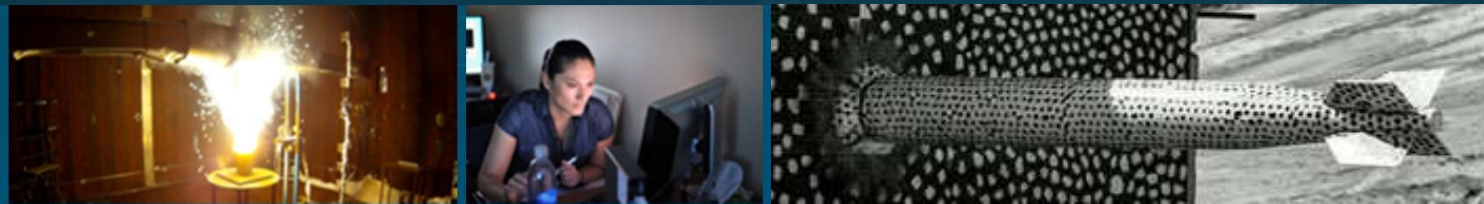


CCC Workshop on Physics & Engineering
Issues in Adiabatic/Reversible Classical Computing



TECHNICAL SESSION III—ARCHITECTURE & HIGH-LEVEL TOPICS
Architectural, Algorithmic, and Systems
Engineering Issues for Reversible Computing



Wednesday, October 7th, 2020

Michael P. Frank, Center for Computing Research

Approved for public release, SAND2020-10467 C



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

2

Workshop Overview



	Day 1 (Mon. 10/5)	Day 2 (Tue. 10/6)	Day 3 (Wed. 10/7)	Day 4 (Thu. 10/8)	Day 5 (Fri. 10/9)
START TIME (US PDT)	TECHNICAL SESSION I: FUNDAMENTAL PHYSICS	TECHNICAL SESSION II: DEVICE & CIRCUIT TECHNOLOGIES	TECHNICAL SESSION III: ARCHITECTURE & HIGH-LEVEL TOPICS	FIRST DAY OF WORKING MEETINGS	SECOND DAY OF WORKING MEETINGS
8:30 am	Workshop Intro	Keynote: Ed Fredkin	M. Frank, G. Snider, N. Yoshikawa, H. Thapliyal, R. Wille	(9a.) Day 4 Intro.	(9a.) Day 5 Intro.
9:20 am	Mike Frank	Mike Frank		Outbriefs from Breakouts	Outbriefs from Re-Breakouts
10:00 am	Norm Margolus	Sarah Frost-Murphy Jie Ren			
10:20 am	Early Break	Early Break		Early Break	Early Break
10:50 am	Neal Anderson	Kevin Osborn	Erik Demaine	Concordance Discussion #1	Concordance Discussion #2
11:10 am	Subhash Pidaparthi	Ralph Merkle	Robert Glück		
11:30 am	Karpur Shukla	Joe Friedman	Erik DeBenedictis		
11:50 am	Panel / Q&A	Panel / Q&A	Panel / Q&A	Late Break	Late Break
12:10 pm	Late Break	Late Break	Late Break		
12:40 pm until...	Physics Breakout	Techno. Breakout	Arch./HL Breakouts	Re-Breakouts	Final Breakout & Concordance

Abstract Text



The reversible computing paradigm has long-term implications that extend beyond the device and circuit levels, eventually impacting all aspects of computer design. Critical to the continuing development of the reversible approach will be the appropriate consideration, by system designers, of various overheads, scaling relations, and design trade-offs that come into play. In this talk, we survey the various issues that impact scaling, and briefly review some architectural, algorithmic and higher-level techniques that can help to address them.

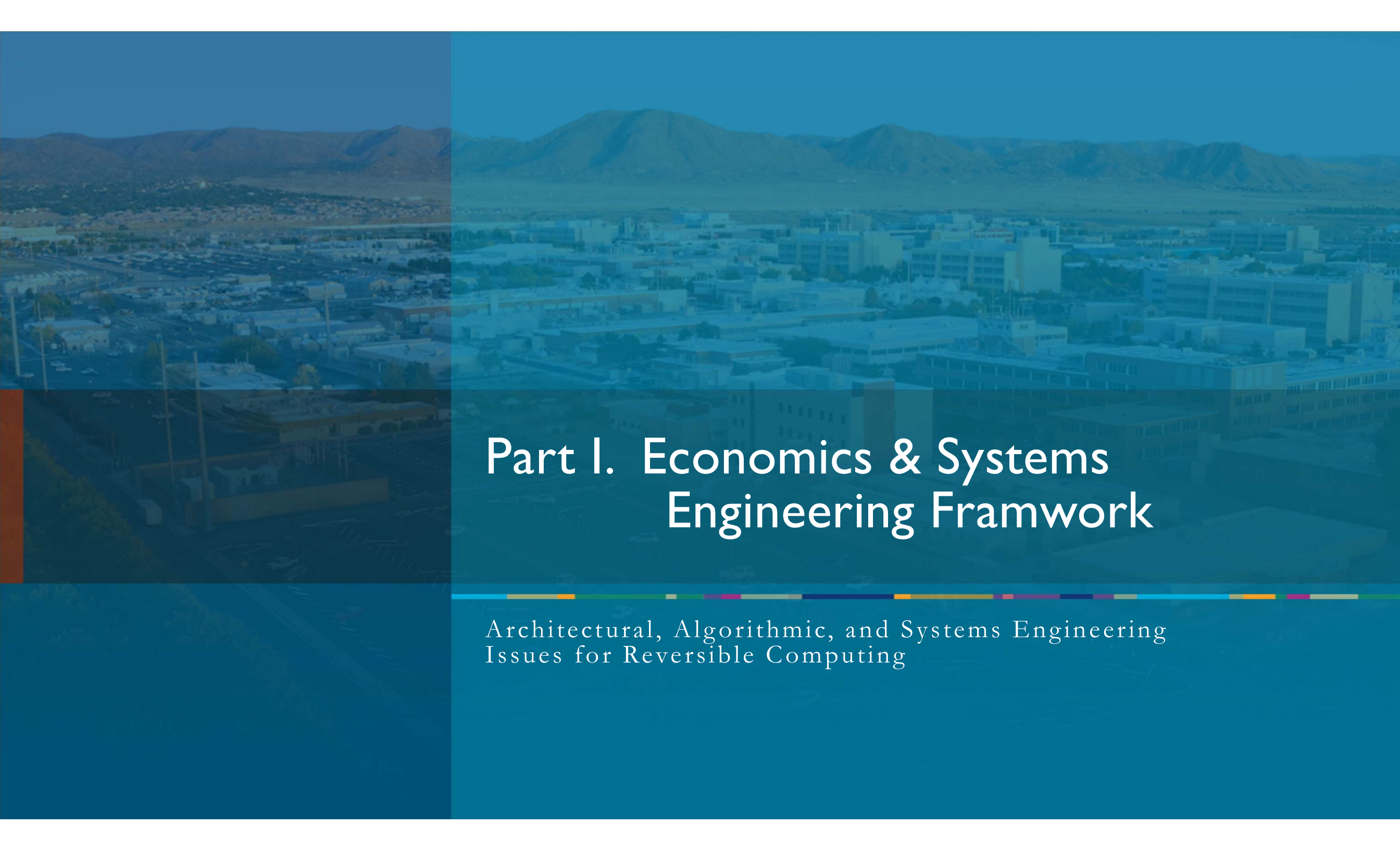
Outline of Talk



Architectural, Algorithmic, and Systems Engineering Issues for Reversible Computing

- I. Basic economic/systems engineering framework
 - Cost & efficiency metrics
- II. *Classic* computational complexity theory for reversible computing...
 - ...and, why traditional complexity theory is inadequate for practical engineering purposes!
- III. *Physical* theory of computing complexity
 - Physically-realistic models of computation.
 - Scaling advantages of reversible computing in physical computing models.
 - Practical tradeoffs and scaling relations for reversible computing systems.
- IV. Elements of reversible architectures and languages
 - Why new HDLs are needed for reversible architecture.
 - Some concepts of reversible instruction-set architectures (ISAs).
 - Some concepts of reversible programming languages.
- V. Conclusion





Part I. Economics & Systems Engineering Framework

Architectural, Algorithmic, and Systems Engineering
Issues for Reversible Computing

Motivation from Economics / Systems Engineering



In general, *efficiency* η of any process can be defined as the amount P of some valued *product produced* by the process, divided by the amount C of *cost consumed* (in terms of resources, or dollars) by the process.

$$\eta = \frac{P}{C}$$

- For a computing system,
 - P can be amount of useful *information processing performed* (e.g., number of operations) by the system over its operating lifetime, and
 - C can be expressed the sum of manufacturing (& deployment) costs, plus operating costs over the system lifetime.
- We can also annualize the costs, in terms of, e.g. time-amortized manufacturing cost.
 - More sophisticated variations that account for net present value of future returns, depreciation curves, *etc.*, not considered here.
- Operating costs largely amount to *energy-proportioned costs*: $C_{\text{oper}} = c_{\text{en}} \cdot E_{\text{oper}}$
 - c_{en} = operating cost per unit of energy dissipated; E_{oper} = energy dissipated during a given period of operation.

$$C = C_{\text{tot}} = C_{\text{mfg}} + C_{\text{oper}} \quad (\text{may be amortized})$$

We can thus reduce the efficiency formula $\eta = P/C_{\text{tot}}$ for computing to the form at right:

- E_{op} = Energy dissipated due to *one* primitive device operation (or by one primitive device in time t_d).
- $c_{\text{dev},t}$ = Amortized manufacturing cost per primitive device per unit time t .

$$\begin{aligned} \eta &= \frac{1}{c_{\text{en}} \cdot E_{\text{op}} + c_{\text{dev},t} \cdot t_d} \\ &= \frac{1}{E_{\text{op}} t_d \left(\frac{c_{\text{en}}}{t_d} + \frac{c_{\text{dev},t}}{E_{\text{op}}} \right)} \end{aligned}$$

Some observations from this equation.:

- There are *diminishing* efficiency returns from decreasing *either* E_{op} or the $c_{\text{dev},t} \cdot t_d$ term in isolation
 - \therefore Continuing to push non-reversible technologies will ultimately reach a dead end!
- Note that if *both* E_{op} and $c_{\text{dev},t}$ were decreased by $N\times$, overall efficiency would be increased by $N\times$. (All else being equal.)
- Decreasing $E_{\text{op}} \cdot t_d$ (dissipation-delay product, DdP) is *often* (but not always!) a win.
 - E.g., in scenarios where total lifetime cost of operation starts out very heavily energy-dominated, total cost can be reduced by lowering E_{op} , *even* in cases where $E_{\text{op}} t_d$ stays the same, or even increases somewhat!
- However, at any given per-device cost, decreasing $E_{\text{op}}(t_d)$ (dissipation as a function of delay) for any given delay value t_d is *always a win*.
 - Thus, this will be our focus in future work.



Part II. Classic Complexity Theory for Reversible Computing

Architectural, Algorithmic, and Systems Engineering
Issues for Reversible Computing

Classic Results in Reversible Computing Complexity

NOTE: All of the below results are based on the *classic* theory of computational complexity, which *ignores* realistic physical constraints!

Bennett 1973:

- **RTIME**(T) = **TIME**(T) - Reversible time equals irreversible time.
- **RST**(ST, T) \supseteq **ST**(S, T) - Relation between reversible and irreversible joint space-time complexity classes.
 - Up to $\Theta(T)$ space overhead factor to store intermediate “garbage” results before decomputing.

Bennett 1989:

- **RSPACETIME**($S \cdot T^{\overbrace{\log_2 3}^{1.59}} \cdot \log T$) \supseteq **ST**(S, T). – Beats previous by $T^{0.41} / \log T$.
 - “Pebble Game” algorithm.

Lange et al. 1997:

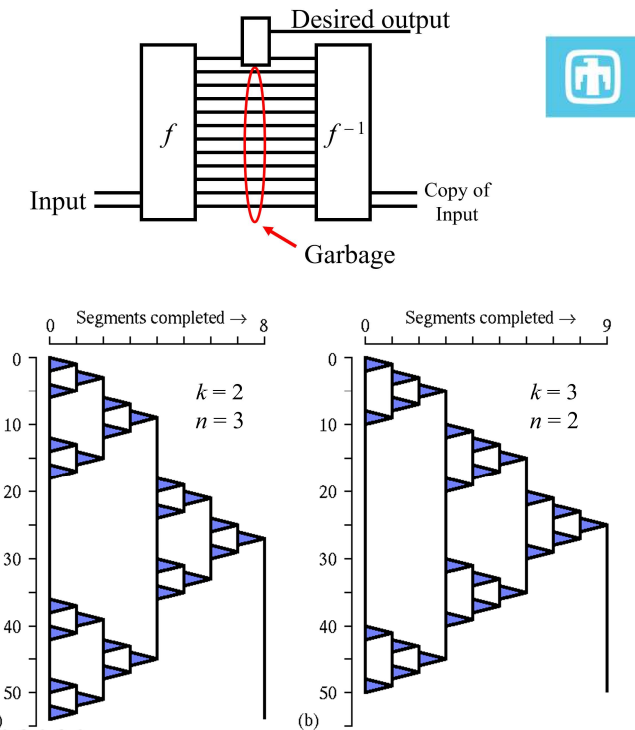
- **RSPACE**(S) = **SPACE**(S) – Reversible space equals irreversible space!

Frank & Ammer 1997:

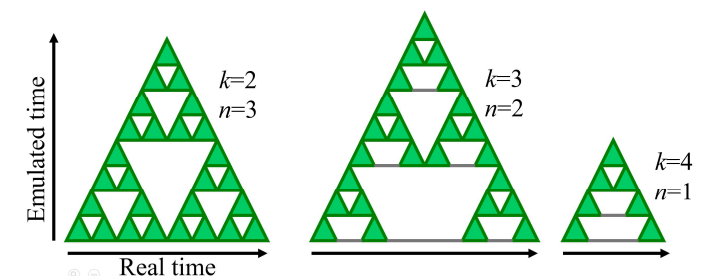
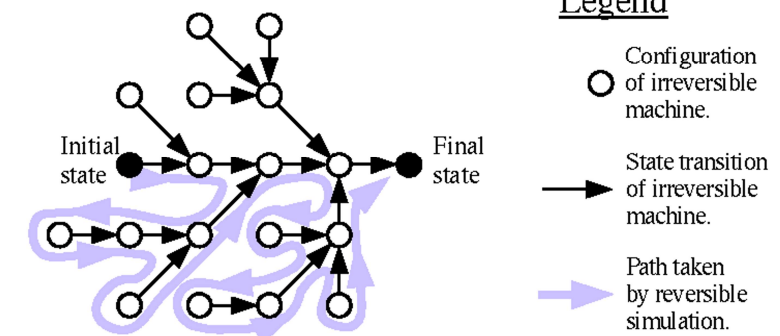
- $\exists O: \mathbf{RST}(S, T)^O \neq \mathbf{ST}(S, T)^O$ for a reversible oracle O . (Black-box function.)

Frank 2002:

- Parallel variant of Bennett ‘89 algorithm improves its spacetime efficiency.



(Lange, McKenzie, Tapp ‘97)



But... The entire traditional field of computational complexity theory turns out to be *fatally inadequate* for our present purposes!



In the sense that, it *is not useful* at all for making any kind of reliable determination regarding whether reversible machines are more or less efficient than irreversible ones in practice, *because...*

Traditional computational complexity theory *ignores* important fundamental physical constraints on computation!

Such constraints include:

- The speed-of-light limit on information propagation velocity, which impacts communication latencies.
- Various limits on information *density* (from general relativity and field theory, and much-closer practical limits).
- Limits on information *flux* densities (follow from the above two limits).
- Landauer's Principle, which sets a lower bound on entropy generation from irreversible computation.
- Quantum limits on rate of state change as a function of invested energy.

When realistic physical constraints such as the above are included in our theoretical framework for modeling computation, we find that models that include a reversible computing capability *strictly dominate* ones that do not!

- That is, they exhibit strictly ***greater*** asymptotic scaling of efficiency (*i.e.*, strictly ***reduced*** real-world *physical* cost or complexity) on a wide range of problems!
 - And, this is certainly not at all obvious if you consider only the results from traditional complexity theory.



Part III. Physical Complexity Theory for Reversible Computing

Architectural, Algorithmic, and Systems Engineering
Issues for Reversible Computing

Physical Complexity Theory – Overview of Some Key Results



Frank '97: Assuming just classic adiabatic ($1/t$) scaling of dissipation with delay, if leakage is negligible, and assuming information density/flux limits:

- Reversible performance per unit power consumption, or per area is **unboundedly greater** than irreversible.
 - Can even be practical if manufacturing cost can be made negligible, or if we can amortize it over an unboundedly-large system lifetime.
- For cases where algorithmic overheads are negligible, and the mass and volume per device is fixed,
 - Reversible performance per area scales up with order \sqrt{d} , the **square root** of the thickness d (or mass per area) of the machine.

Frank '99: Shows performance advantages per unit mass *even when the cost of energy is negligible* (or another way of saying this is, even if the power budget is unlimited):

- With the same assumptions as above, reversible performance advantage **per unit mass** scales by as much as order $N^{1/18}$ (where N is number of processing elements) for some (communication-limited) problems.
 - Note, showing an advantage per-mass makes the prospect of an overall cost-efficiency advantage more plausible.
- If a perfect *ballistic* reversible technology were invented (negligible dissipation at finite speed), the performance boost per unit mass improves to order $N^{1/9}$.
 - E.g., for a machine with 10^{18} micron-scale processing elements, performance gain from reversibility on some problems is boosted by 100×.
 - Note that this is a strong statement! We are saying that this performance boost holds *no matter what architecture is chosen* for the irreversible machine, and *even if energy is free!*

Performance per-area scaling with machine thickness



Frank & Knight 1997, doi:[10.1088/0957-4484/9/3/005](https://doi.org/10.1088/0957-4484/9/3/005)

Assumptions of this simple analysis include:

- Classic adiabatic ($E_{\text{diss,op}} \propto 1/t$) scaling
- Fixed operating temperature
- Constant volume and mass per device
- Bounded entropy flux density F_S
- No algorithmic overheads for reversibility

Later, we will discuss the impact of considering the algorithmic overheads of reversibility.

- Spoiler: Reversible computing still wins!

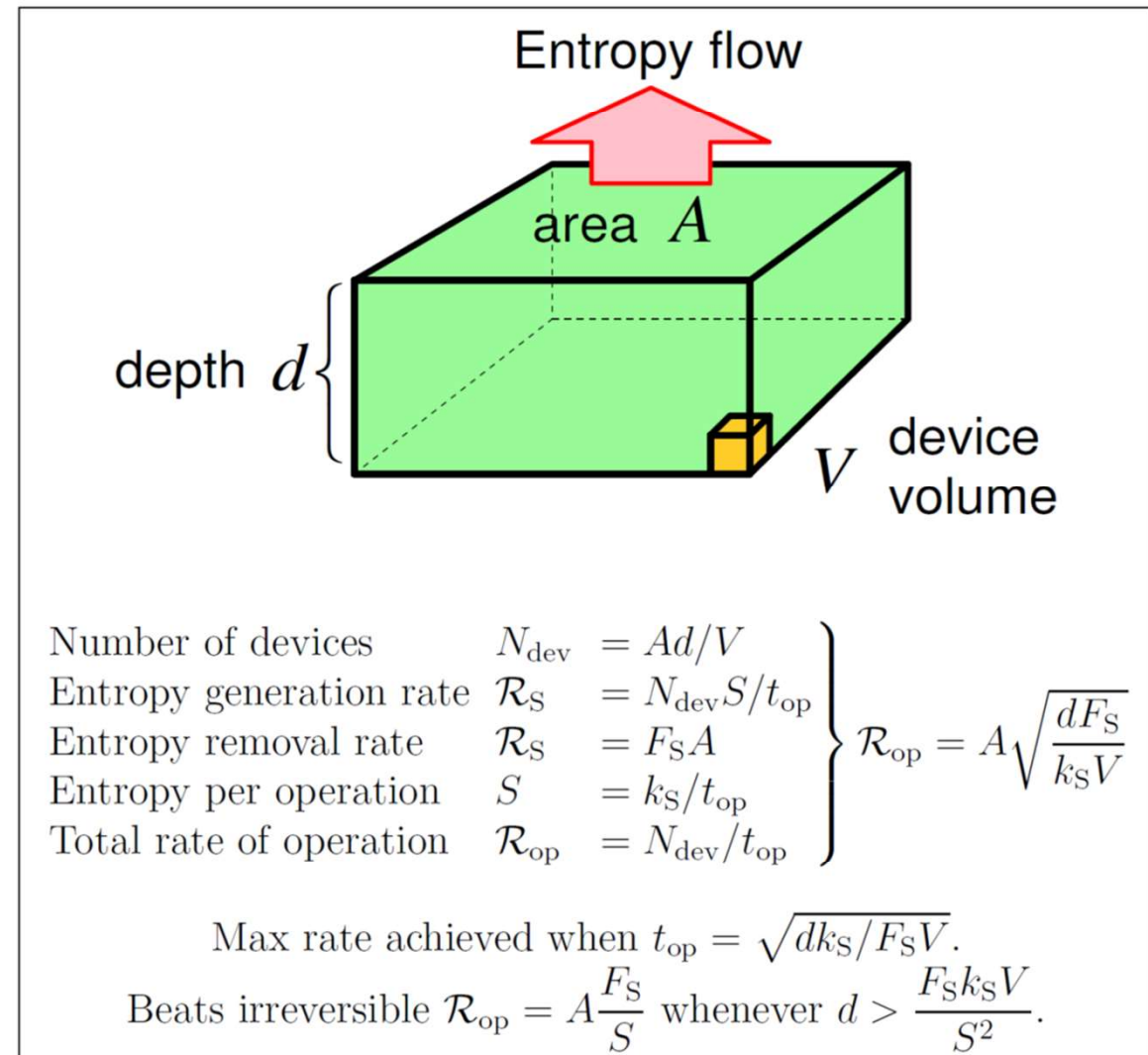


Figure 6.1: Speed limit for reversible machines of minimum-surface area $\Theta(A)$ and thickness $d \lesssim A^{1/2}$. The maximum rate of computation scales as $\Theta(A\sqrt{d})$.

Performance per-mass scaling

Frank 1999, <https://dspace.mit.edu/handle/1721.1/9464>

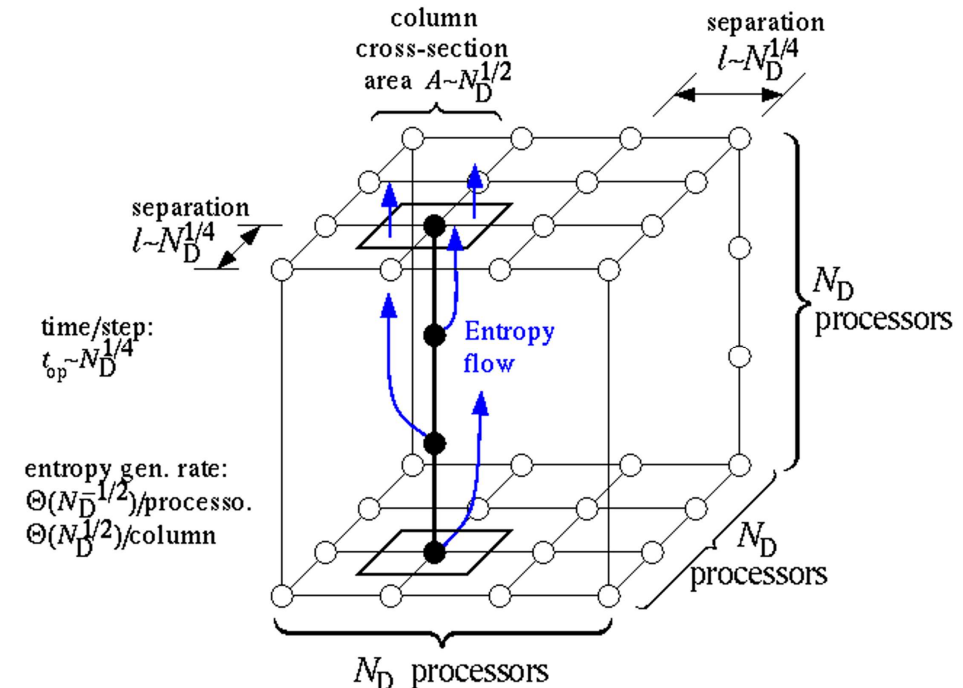
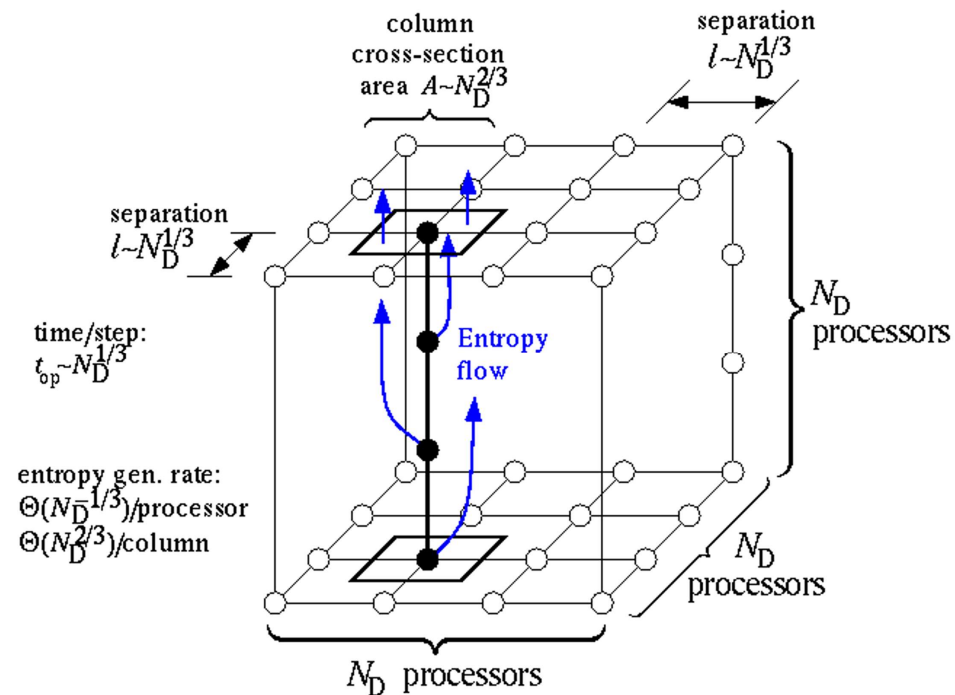


If the application requires *no* communication between processing elements, and if energy is free, no (asymptotically increasing) speed advantage can be shown for reversible computing in that case.

- Because we can spread out processing elements arbitrarily far, and feed them arbitrarily large amounts of power.

Thus, showing a scalable performance-per-mass boost with no energy constraints requires considering applications that require frequent communications between PEs.

- If only *local* communication is needed per step (below), advantage on a 3D problem is only $N_D^{1/12} = N^{1/36}$.
 - But, if the required communication distance is $N_D^{1/2}$, advantage improves to $N^{1/18}$.
- But, for an ideal *ballistic* machine, the reversible advantage even on the *local* 3D problem improves to $N^{1/9}$!



Accounting for Nonidealities

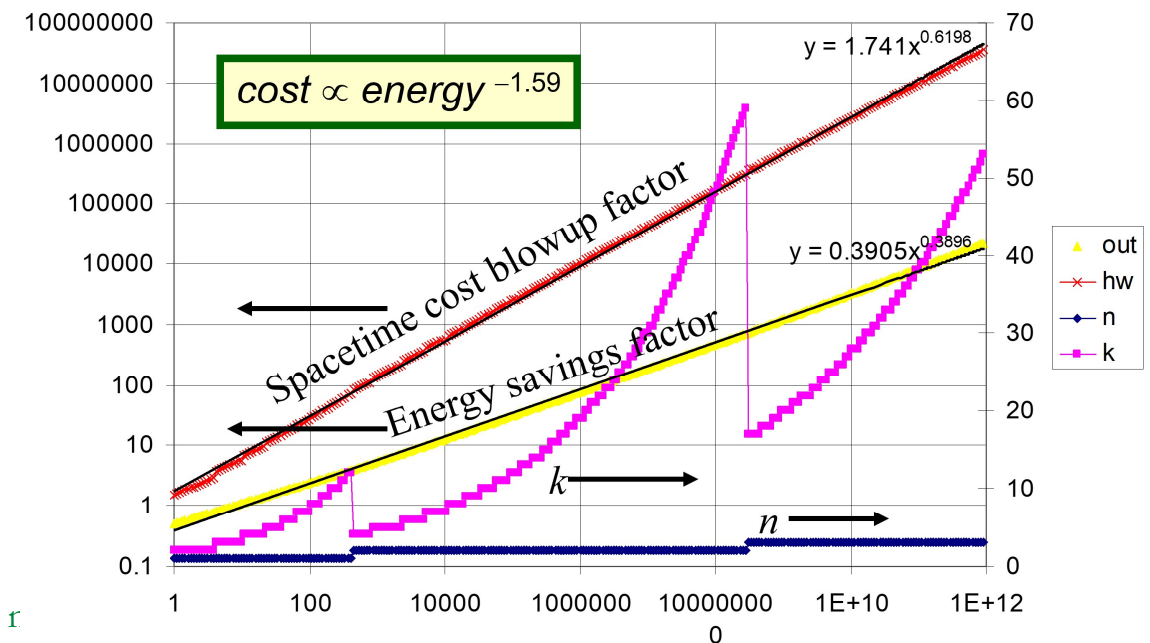
The above analyses assumed that leakage can be engineered to be as small as necessary for it not to be limiting (which may be an OK assumption for some technologies) and negligible algorithmic overheads (which may be an OK assumption for *some* problems).

- But, can we still show an advantage even when making more pessimistic/realistic assumptions?
 - Answer is yes!

Even for worst-case problems, we can always at least still use the “Frank ‘02” algorithm mentioned earlier.

- And, even better general “reversiblization” algorithms may yet be discovered in the future.
 - Then, as the technology is improved, and leakage is reduced, we can adjust the parameters of the algorithm to minimize the total cost (including both energy and spacetime associated costs).
- We find that we can reduce total lifetime *system* cost by any factor of N if we just reduce leakage by $\sim N^{2.56}$ and time-amortized per-device manufacturing cost by $\sim N^{1.59}$.
- Example: To achieve an $N = 1,000 \times$ *overall* efficiency boost, reduce leakage by $47.8\text{M} \times$ and mfg. cost/device by $59,000 \times$.
 - Ambitious but doable!! This gives us a way forward, where otherwise there is r

Worst-Case Energy/Cost Tradeoff (Optimized Bennett-89 Variant)





Part IV. Elements of Reversible Architectures and Languages

Architectural, Algorithmic, and Systems Engineering
Issues for Reversible Computing

Elements of Reversible Architectures and Languages



In this section, we discuss/review:

- Why novel features are needed in hardware description languages to support digital design of reversible architectures.
- Why new algorithms are needed for reversible machines.
 - Here we mean low-level hardware algorithms initially, but eventually also higher-level application algorithms.
- Some concepts of reversible instruction set architectures.
- Some concepts of reversible high-level programming languages.

Later speakers in this session will elaborate on some of these themes.

New Features Needed in Hardware Description Languages and Design Tools to Support Adiabatic/Reversible Design



Reversible functional units will in general come with *preconditions for reversibility* that are required to be satisfied in order for their operation to actually be logically and physically reversible.

- In general, these can include timing constraints, constraints on the logical relationship between the initial states of different input signals and internal data values, and constraints on the relationship between the functional unit and others that may try to control the same output node at the same time.
 - E.g., in general, two units should not try to actively drive the same node to two different values simultaneously (true even standardly).
- It would be highly desirable for the HDL to support expression of such constraints, and for the simulation environment to support checking of satisfaction of the constraints at the level of discrete simulations.
 - To the extent that *compile-time* constraint checking is feasible, this should also be supported.
 - E.g., functional units can guarantee *postconditions*, which can be compared against preconditions of units receiving their output.
 - Automated compile-time verification (e.g., by theorem proving) that certain invariants are maintained by a given sequential design.

Logic synthesis tools should of course be able to synthesize reasonably efficient reversible hardware algorithms based on known techniques,

- and should be capable of optimizing appropriate tradeoffs between e.g., degree of reversibility and hardware overhead,
 - given information about the relative economic weight of energy and hardware costs.

Why new algorithms are needed for reversible machines!



Briefly, because in general *the most efficient reversible algorithm for a given problem could have a very different structure from the most efficient irreversible algorithm for the same problem.*

- In other words, no *automated* “reversiblization” technique can be expected to be able to discover the best (or even a “good enough”) reversible algorithm for a given problem, even assuming that the best (or a “good enough”) irreversible algorithm is already provided.
- Thus, serious research into reversible algorithms (at both the hardware and software level) is needed!
 - Today’s speakers will mention some of the work that’s been done in this area.

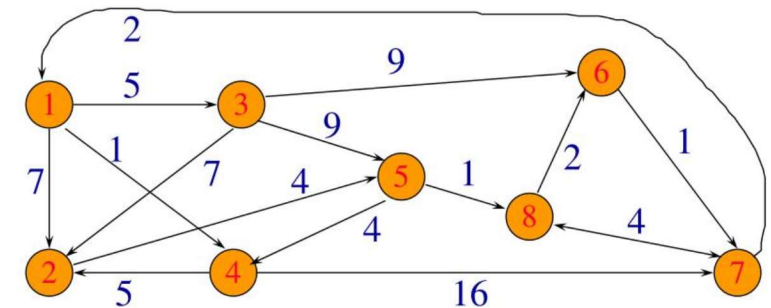
One example: *All-Pairs Shortest Paths* graph problem →

- One good *irreversible* algorithm is Floyd-Warshall.
 - Space $\Theta(n^2)$, time $\Theta(n^3)$.
- Simple, direct reversiblization seems to require space $\Theta(n^3)$!
 - However, there is an alternative algorithm (with a very different structure) for which a reversible version takes only $\Theta(n^2 \log n)$ space and $\Theta(n^3 \log n)$ time.

At the hardware level, even very basic problems such as finding the most efficient reversible n -bit adder design remain unsolved!

- Although this is unsurprising, since even for *irreversible* logic, lower bounds on circuit complexity for Boolean functions are hard to find.

- Given an n -vertex directed weighted graph, find a shortest path from vertex i to vertex j for each of the n^2 vertex pairs (i,j) .



Early History of Reversible Processor Architectures

Ed Barton (Student of Ed Fredkin; MIT class project, 1978)

- Conservative logic processor, with garbage stack

Andrew Ressler (Student of Ed Fredkin; MIT bachelor's thesis, 1979; MIT master's thesis, 1981)

- Similar to Barton's design, but more detailed. Paired branches.

Henry Baker (1992)

- Instruction set for a reversible pointer automaton machine.

J. Storrs "JoSH" Hall (1994)

- Retractable-cascade-based PDP-10-like architecture.

Carlin Vieri (MIT master's thesis, 1995)

- Early Pendulum ISA, irreversible (Verilog) implementation, full RTL-level detail.

Frank & Rixner (MIT VLSI class project, 1996)

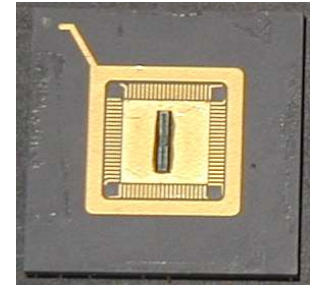
- **TICK**: Irreversible VLSI schematics & layout implementing an 8-bit subset of Pendulum ISA, plus paired branches.

Frank & Love (MIT VLSI class project, 1996)

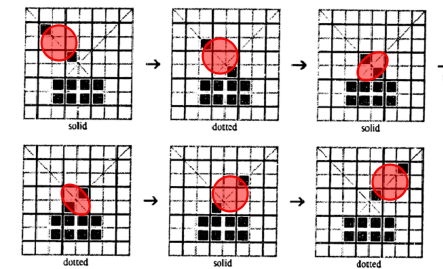
- **FLATTOP**: Adiabatic VLSI design of programmable reversible gate array implementing Margolus' BBMCA (Billiard-Ball Model Cellular Automaton).

Vieri (MIT Ph.D. thesis, 1999)

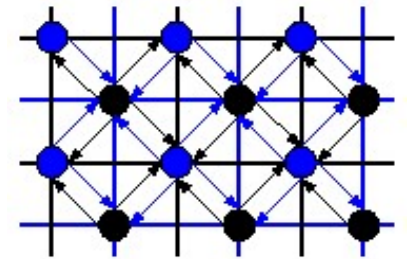
- **PENDULUM**: Fully adiabatic 32-bit VLSI implementation of PISA with paired branches.



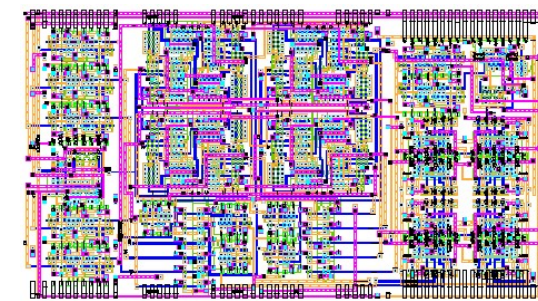
TICK (MIT '96)



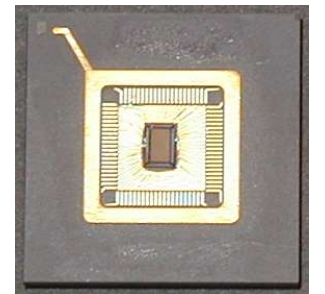
Margolus '88 BBMCA



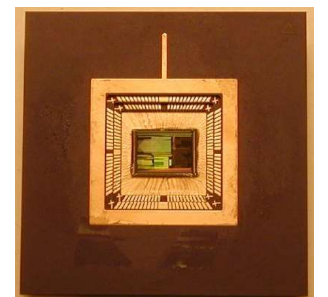
FLATTOP PE mesh



FLATTOP unit cell



FLATTOP (MIT '96-98)



PENDULUM (MIT '95-99)

Some Reversible ISA Concepts

Early reversible architectures had a “garbage stack,”

- But this turns out not to be necessary.

“Non-expanding” arithmetic/logical operations:

- No particular overhead to do these reversibly in-place

“Expanding” arithmetic/logical operations:

- One reversible method: XOR result into destination register
- Programmer manages garbage data

Paired branches:

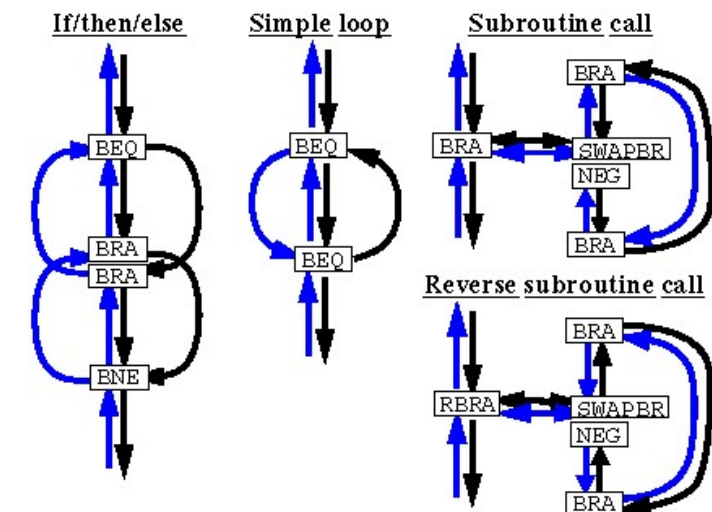
- Innovation of Ressler (‘79-81), reinvented by Frank ‘95
- Avoids accumulation of garbage from control-flow operations
- Changes control-flow semantics just a bit
 - But, fairly ordinary forms of structured control flow are still possible!

Example PISA-type instructions:

“Nonexpanding” arith./logic:			“Expanding” arith./log.:		
NEG	ra	(ra = -ra)	ANDX	ra,rb,rc	(ra ^= rb&rc)
ADD	ra,rb	(ra += rb)	ANDIX	ra,rb,imm	(ra ^= rb&[imm])
ADDI	ra,imm	(ra += [imm])	ORX	ra,rb,rc	(ra ^= rb rc)
SUB	ra,rb	(ra -= rb)	SLLX	ra,rb,imm	(ra ^= rb<<imm)
XOR	ra,rb	(ra ^= rb)	SLTX	ra,rb,rc	(ra ^= (rb<rc)?1:0)
RL	ra,imm	(ra <=< imm)	SRAX	ra,rb,imm	(ra ^= rb>>imm)

Branch instructions:		
BEQ	ra,rb,off	(if ra=rb, BR+=off*dir)
BGEZ	ra,off	(if ra>=0, BR+=off*dir)
BGTZ	ra,off	(if ra>0, BR+=off*dir)
BRA	loff	(BR += loff*dir)
RBRA	loff	(dir=-dir, BR+=loff*dir)
SWAPBR	ra	(ra <-> BR)

Some reversible control flow structures:



Some Reversible High-Level Language Concepts



Fairly standard sorts of high-level language semantics can be supported.

- Some new constraints for correct reversible operation of control constructs...
 - *E.g.*, don't change the truth value of an **if** condition within the body

The R (or \mathbb{R} , pronounced “yar”) compiler was completed in 1999.

- Procedural language; vaguely C-like semantics.

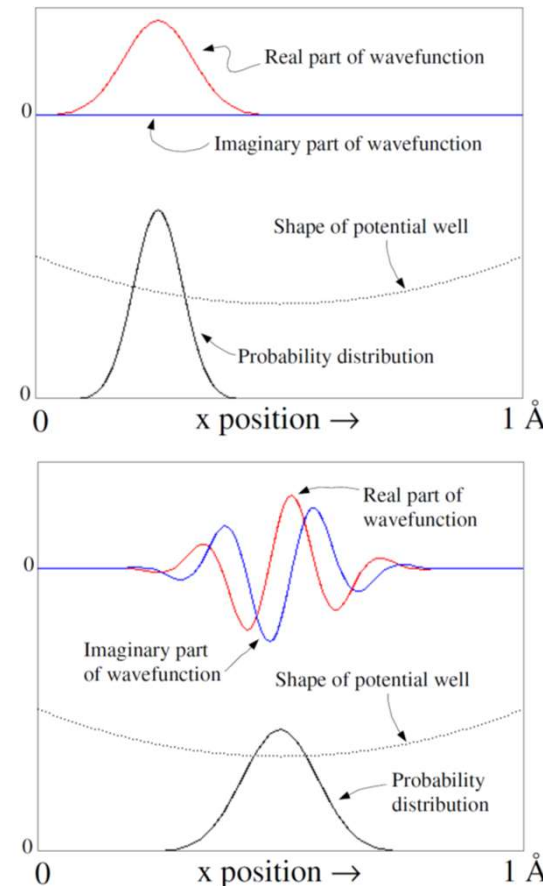
An example application was demonstrated:

- A *garbage-free* reversible program to simulate the (1-D) Schrödinger wave equation of quantum mechanics!

Kernel of Schrödinger simulator in \mathbb{R} :

```
;; This subroutine updates one of the two waves, using the other.
(defsub halfstep (dest src)
  (let (e <- epsilon)
    (for i = 0 to 127
      (let (d <-> (dest _ i))
        (d += ((alphas _ i) */ (src _ i)))
        (d -= (e */ (src _ ((i + 1) & 127))))
        (d -= (e */ (src _ ((i - 1) & 127))))))))

;; Main program, goes by the name of SCHROED.
(defmain schroed
  (for i = 1 to 1000 ;Enough time for electron to fall to well bottom.
    ;; Take turns updating the two components of the wave.
    (call halfstep psiI psiI)
    (rcall halfstep psiI psiI))
```



The “R” reversible programming language



Features:

- Parenthesis-based syntax, for easy parsing.
- Simple C-like procedural semantics.
- Permits coding efficient reversible algorithms.
- The R language compiler:
 - Written in Common Lisp.
 - Targets PISA assembly code.

Example R language constructs

Control flow:

```
(if condition then
  statement ... )
(for var = start to end
  statement ... )
(defsub subname (arg1 arg2 ... )
  statement ... )
(call subname arg1 arg2 ... )
(rcall subname arg1 arg2 ... )
```

Data manipulation:

```
(let (var <- val)
  statement ... )
(place ++ )
(place += value)
(place -= value)
(place ^= value)
:
```

Expressions:

```
(val1 + val2)
(val1 - val2)
(val1 & val2)
(* address)
(array _ index)
:
```

Data declaration:

```
(defword name value)
(defarray name
  value0 value1 ... )
```

Output:

```
(printword val)
(println)
```



Part V. Conclusion

Architectural, Algorithmic, and Systems Engineering
Issues for Reversible Computing



At the most fundamental level, the basic economic/systems-engineering rationale for reversible computing is extremely easy to state:

- Assuming only that energy has a nonzero minimum cost, continued scaling of time-amortized manufacturing cost per-device *cannot yield substantial reductions* in total lifetime system cost of ownership for given computational workloads (or equivalently, increases in the scale of workloads executable within given lifetime cost budgets) *unless energy dissipation per operation also continues to decline commensurately*. (Which eventually requires using RC.)

Some additional key points regarding scaling of reversible machines:

- As 3-D stacked fabrication processes become more viable, performance *per area* of even *classic* adiabatic computing scales up with the *square root* of the number of layers L . (Versus no scaling possible at all for conventional logic!)
 - As soon as manufacturing cost per-area for any given L is brought below proportional to \sqrt{L} , this becomes an economic win for that value of L .
- *Physical computing theory* also allows us to *prove* that, ***even if energy itself were free***, adiabatic machines still would have better asymptotic *performance-per-mass* scaling than irreversible machines, assuming only that the application requires frequent inter-processor communication, and that leakage can also be scaled down accordingly.
 - And, this is still the case even when accounting for algorithmic overheads on worst-case problems!
- In general, the scaling advantages of reversible computing in all of these scenarios would become dramatically even better, if we could invent ideal *ballistic reversible* technologies exhibiting negligible dissipation at useful, finite speeds.
 - And, perhaps this can be done by using concepts such as exponential adiabaticity, superadiabaticity, or shortcuts to adiabaticity (STA).

Eventually, the requirement for an increasing degree of reversibility will pervade computing at all levels, from microarchitecture, to processor design, to the design of high-level languages and algorithms.

- The required changes are generally straightforward, but substantive, and require new research at all these levels!
 - Thus, reversible computer science is an important field for the future, even apart from its possible applications in quantum computing.