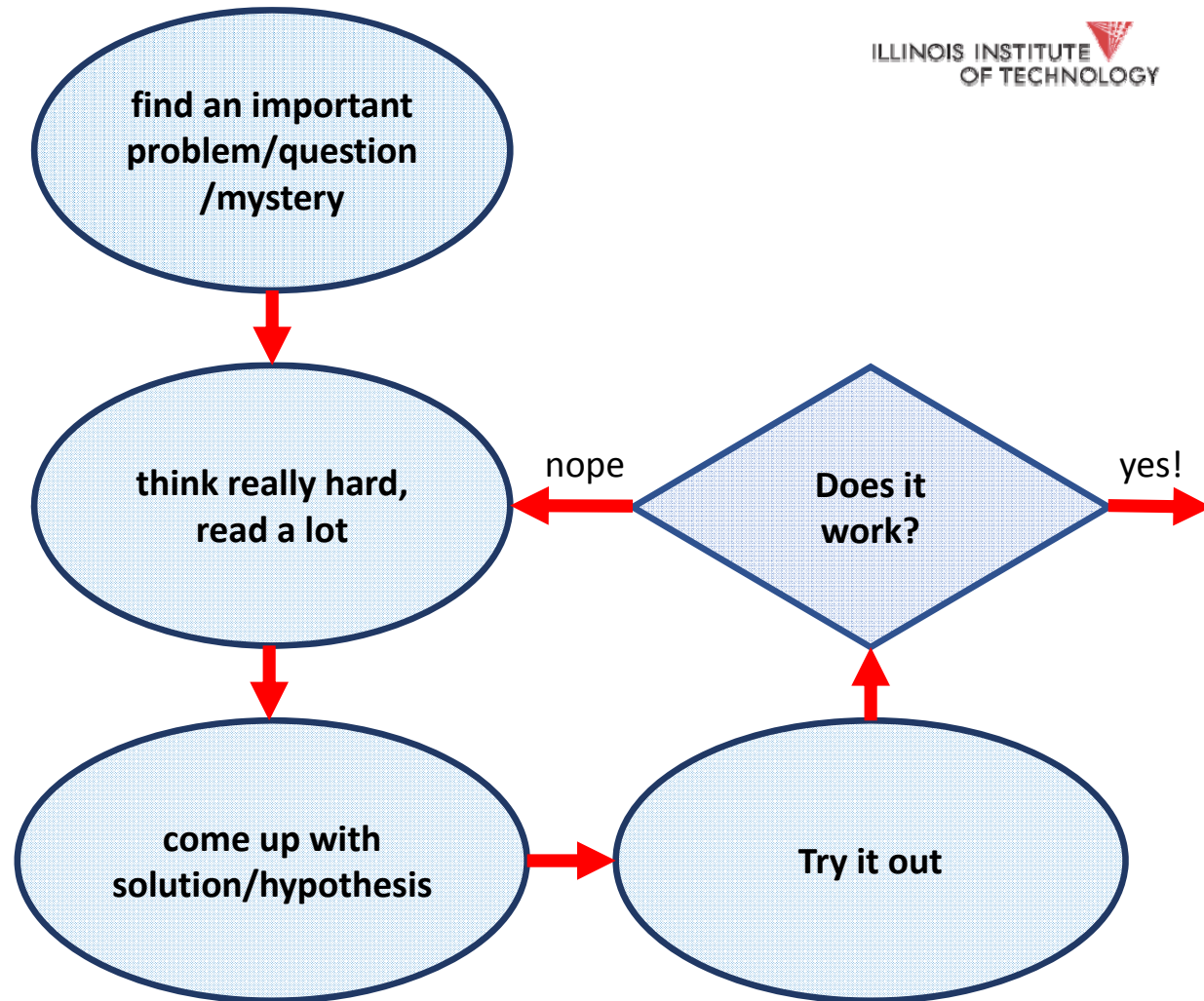


Undergraduate Research @ IIT

Kyle C. Hale

What is research?



ASC: Automatically Scalable Computation

Abstract

We present an architecture designed to transparently and automatically scale the performance of sequential programs as a function of the hardware resources available. The architecture is predicated on a model of computation that views program execution as a walk through the enormous state space composed of the memory and registers of a single-threaded processor. Each instruction execution in this mode system from one state to a deterministically-determined state. We can parallelize such execution by partitioning the complete path and speculatively each partition in parallel. Accurately partitioning a challenging prediction problem. We have implemented a system in an architectural simulator of an x86 processor a collection of state predictors and a mechanism for sequentially executing threads that explore potential state execution path. We demonstrate that we can achieve 256 speedup on 1024 cores while running an sequential program.

The primary design challenge in realizing this architecture is accurately predicting points that partition a trajectory. We break this challenge into two parts: (1) recognizing states from which accurate prediction is possible and will result in useful speedup, and (2) predicting future states of the system when the current state of execution is recognized as one from which prediction is possible.

230

A. M. TURING

[Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

rejoice! and tell the whole world!

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbersome technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable

9/6/2019

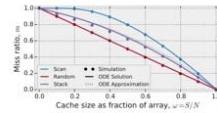


Fig. 2. Comparison of ODE solution (—) vs. simulation (•) on random replacement for several access patterns.

These equations eliminate variables A and B , simplifying the analysis. The initial values for all variables are zero, so $D(0) = E(0) = 0$ and (by substitution) $H'(0) = D'(0)$ and $F'(0) = m/N$. (Age zero simply denotes infinitesimally small ages.) These initial values mean that there are no hits or evictions initially, so short reuse distances hit, and, since small ages are more common, evictions are also more likely.

Scanning pattern: To solve for the cache's miss rate on a scanning pattern with random replacement, we first observe that (i) all lines hit at age N if not evicted, (ii) no hits occur at ages less than N , and (iii) all evictions occur at ages less than N . Thus for ages $0 < a < N$, $H'(a) = 0$ and

$$E'(a) = \frac{m}{N} E'(0) \Rightarrow E = 1 - e^{-m/N a} \quad (12)$$

The right-hand side follows from initial conditions. We can now compute the miss rate m . Since no evictions occur beyond age

$$N, \quad m = \int_0^N E'(a) da = E(N) - E(0) = 1 - e^{-m} \quad (13)$$

We solve for m via the product logarithm $W(x) = x \Rightarrow x e^x = x$

shows that the model is, once again, an excellent match with simulation.

Stack pattern: Similar to the scanning pattern, the stack pattern only has hits or evictions at ages $0 < a \leq 2N$. We simplify the hit distribution using the derivation of Eq. 10, $D'(a) = 1 - a$ and $D''(a) = 0$

$$H'' = \frac{E'}{2N} \quad (19)$$

Given the initial conditions, these equations are solved by

$$H' = \frac{1 - e^{-m a}}{2N} \quad \text{and} \quad E = (1 - \frac{m}{2N}) e^{-m a/2} \quad (20)$$

This yields the miss rate

$$m = \int_0^{2N} E'(a) da = 1 + \frac{e^{-m N} - 1}{2mN} \quad (21)$$

Unlike prior patterns, Eq. 21 does not yield an explicit solution for m (it is a transcendental equation). But it can be used to quickly iterate to a fixed-point on m by repeatedly evaluating the right-hand side of Eq. 21. This iteration is far more efficient than iterating on a full discrete model like $\{1, 3, 4, 6, 8\}$.

Although an exact, closed-form solution is not forthcoming for the stack distribution, we can get a closed-form approximation. First, consider that when $N \gg 1$, the miss rate is high $m \approx 1$, the cache is small $\omega \approx 0$, and thus $e^{-m \omega} \approx 0$. Hence

$$m \approx 1 - \frac{1}{2N} \Rightarrow m \approx \frac{1}{2} (1 + \sqrt{1 - 2\omega}) \quad (22)$$

This approximation works well at high miss rates, and we can extend it across the full range by taking its Taylor series around

$$\omega = 0 \Rightarrow m \approx \frac{1}{2} - \frac{\omega}{2} + \frac{\omega^2}{2} - \frac{\omega^3}{2} + \dots \quad (23)$$

Fig. 2. Compare this approximation (—) to 100 iterations of Eq. 21 (•) and simulation (•). It shows that the approximation accurately throughout the range. The small error at 0 comes from round-off error not being.

patterns

is easily extended to model more complex by combining the patterns) as an example of a program scanning y

Suppose that a program accesses m of p accesses scanning an array of y scanning an array of size N . This is a complicated reuse distance distribution that this pattern also does not as yet we will see that the model remains accurate depending on how often it is the second array. With large reuse d

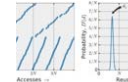


Fig. 3. Compare this approximation (—) to 100 iterations of Eq. 21 (•) and simulation (•). It shows that the approximation accurately throughout the range. The small error at 0 comes from round-off error not being.



Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

I. INTRODUCTION

Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

Real-life document recognition systems are composed of multiple modules including field extraction, segmentation, recognition, and language modeling. A new learning paradigm, called Graph Transformer Networks (GTN), allows such multi-module systems to be trained globally using Gradient-Based methods so as to minimize an overall performance measure.

The main message of this paper is that better pattern recognition systems can be built by relying more on automatic learning, and less on hand-designed heuristics. This is made possible by recent progress in machine learning and computer technology. Using character recognition as a case study, we show that hand-crafted feature extraction can be advantageously replaced by carefully designed learning machines that operate directly on pixel images. Using this approach, we demonstrate that a system can show

Two systems for on-line handwriting recognition are described. Experiments demonstrate the advantage of global training, and the flexibility of Graph Transformer Networks.

A Case for Redundant Arrays of Inexpensive Disks (RAID)

David A. Patterson, Garth Gibson, and Randy H. Katz

Computer Science Division
Department of Electrical Engineering and Computer Sciences
771 Howe Hall
University of California
Berkeley, CA 94720
(gpatt@gaia.berkeley.edu)

Abstract: Increasing performance of CPUs and memories will be quadrupled and replaced by a similar performance increase in I/O. While the capacity of Single Large Expensive Disks (SLED) has grown rapidly, the performance improvement of SLED has been modest. Redundant Arrays of Inexpensive Disks (RAID), based on the magnetic disk technology developed for personal computers, offers an attractive alternative to SLED, promising improvements of an order of magnitude in performance, reliability, power consumption, and scalability. This paper introduces the traits of RAID, gives their relative performance, and compares RAID to an IBM 3390 and a Fujitsu Super Disk.

1 Background: Ramping CPU and Memory Performance
The uses of computers are currently enjoying unprecedented growth in the speed of computers. Gordon Bell said that between 1974 and 1984, the speed of computers improved in performance by 40% per year, about twice the rate of microcomputers (Bell 84). In the following year Bill Joy predicted an even faster growth (Joy 85).

one of magnetic disk technology is the growth in the maximum number of bits that can be stored per square inch, or the bits per inch as a track sense the number of tracks per inch. Called M A I D, for maximal areal density, the "Pursuing an Disk Density" project (Pursuing)

$$M A I D = 10^{10} \text{ bits/in}^2 \text{ (1973/10)}$$

Magnetic disk technology has doubled capacity and halved price every three years, in line with the growth rate of semiconductor memory, and as practice between 1967 and 1979 the disk capacity of the average IBM data processing system more than kept up with its main memory (Stevens 81)

Capacity is not the only memory characteristic that must grow rapidly to maintain system balance, since the speed with which computers and data are delivered to a CPU also determines an ultimate performance. The speed of main memory has kept pace for two reasons (1) the invention of caches, showing that a small buffer can be managed economically to contain a substantial fraction of memory references,

What does it involve?

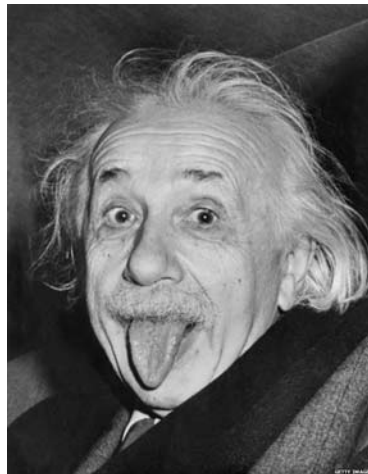
- Thinking (creatively, critically, sometimes wrongly, sometimes rightly) ***a lot**
- Reading ***a lot**
- Learning ***a lot**
- Questioning ***a lot**
- Brainstorming/Collaborating ***a lot**
- Experimenting ***a lot**
- Programming (these days in most fields, esp. CS) ***a lot**
- Writing ***a lot**
- Challenging yourself ***a lot**
- Growing ***a lot**
- Traveling, meeting interesting people ***a lot**
- Joining communities
- Having fun! ***a lot**

What *isn't* research?

- Boredom
- Someone telling you exactly what to do
- Just another 3 credit hours
- Working on a problem someone else has the answer for
- A 9-5 job

Why should I care?

- It's **fun**! You have the **freedom** to explore
- You get to be on the **cutting edge**. Your path to becoming an expert
- You can **contribute something** to the world, creating new knowledge
- Leads to flexible, fulfilling, and well-paying **careers**



Hale



What about research *in CS*?

- One of the most important and game-changing fields currently
- Technology *is everywhere*, it's growing, it ain't slowing down, and it's all **driven by computing**
- A lot of open problems, challenges
- Fast-moving: things change quickly, will never be bored!

CS Research Areas (broadly construed)

- **AI, machine learning, data science**
- **Systems and Networking**
- **Theory**
- **Programming Languages**
- **CS + X**

What does it take to do research?

- **Curiosity:** If you don't ask questions, you won't go exploring
- **Drive + Passion:** You must genuinely *care* if you are to make progress
- **Creativity:** It's not just for the arts! Some of the best research involves new and original ways of rethinking old problems
- **Courage:** You have to learn that **failure is both normal and okay**
 - This is, of course, *unlike* your classes, where you should definitely not fail
- **Dedication:** The best researchers pick a problem and work hard on it

Notice what **I didn't** list...

- **Genius/Talent:** Overrated; it sometimes helps, but **curiosity, passion, and perseverance win out big-time**
- **Straight A's:** The collective sphere of human knowledge couldn't care less about your GPA
 - But *it does* help a professor determine how serious you are about learning...and therefore whether to let you join the lab...so don't ignore them 😊
- **Knowledge:** I'm more interested in working with people who are **eager and willing to learn** more than with those who already know a lot

Career Paths

- **After undergrad -> grad school (PhD)**
 - Yes, it's more school, but it's *not the same*
- **Research Scientist/Engineer**
 - National Labs/Government entities
 - Think Tanks
 - Industry
- **Academic (Professor)**
 - Research + Teaching usually, sometimes just one
- **Consulting**

Opportunities in the Department

- **Informal arrangements:** Generally professors are willing to let you work with them if you're hard-working and motivated
 - e.g., take CS 497 with a prof (independent study)
 - We sometimes have ad hoc summer (or regular semester funding) to pay you for it
- **CS Honors Research Specialization: this is new!** Perform research as you progress through undergrad, write and present thesis at end. Ideally, publish! Talk to us more if you have ??s
- Research Experience for Undergrad – paid (more on this after)
 - At another institution
 - Here at IIT

OK, sounds great; how do I start?

- **Be engaged in class!**
 - Ask *a lot* of questions; **BE CURIOUS**. Don't be afraid to look dumb!
 - Sometimes seemingly obvious questions lead to adventures
 - Do these things and **we will for sure notice!**
- **Get to know your professors**
 - We love talking about research! (we're nerds)
 - Don't be intimidated, we're just people
 - **We will make time for you**
- Find out what you're passionate about, and **approach one of us about doing research**



Important questions in CS

Systems (not exhaustive)

- How to deal with explosion of data (“big data”)?
- How to make extreme amounts of data easy to store, easy to analyze, and easy to compose *efficiently*? -> database systems
- How to make our systems secure?
- How to tame complexity of systems?
- How to design better chip architectures to meet today's challenges?
- How to handle growth of heterogeneous hardware?
- How to make systems *scalable*? (1000s of cores, 1000s of machines)
- How to make systems reliable in the face of failure?

Theory (not exhaustive)

- How to design efficient algorithms for large swaths of (changing) data?
- How to use algorithms to predict behavior and inform decisions?
- Improving computational efficiency across the board (e.g. graph isomorphism, SAT, bin packing, scheduling, etc.)
- $P = NP$? (Just how hard are certain classes of problems? Can we do better?)
- Someone else would know better 😊

AI + Machine Learning + Data Science (not exhaustive)

- How to build (and understand) intelligent machines
 - That have same capabilities of human intelligence: strong AI
 - That will help make our lives better: increase pattern matching, insight abilities of machines
- How to help machines make insights on large amounts of data?
- How to help machines effectively interact with and understand humans? e.g. Natural language processing, knowledge representation, automated reasoning, problem solving
- How to help machines understand the world and interpret data? e.g. computer vision, knowledge representation, etc.
- How to make sure they AI makes positive (and ethical) impact on humanity
- How to make intelligent machines efficient?
- Transparent machine learning: how do we understand *why* and *how* a machine did what it did?
- Are thinking machines conscious? How is our brain different? Philosophy of mind

Programming Languages and Compilers (not exhaustive)

- How do we make it easier to write efficient programs?
- How do we make compilers generate better code?
- How do we automatically port code from one architecture to another?
- What are the right abstractions to current (and new) machines?
- How to create languages that prevent (or at least discourage) bugs?
- How to make programs more secure?
- How to verify that a program does what we think? (verification)
- How to apply tools like machine learning to compilation and program verification?