Branch Mispredictions in Quicksort

K. Kaligosi[1] C. Martínez[2] P. Sanders[3]

[1]Max-Planck-Inst., Germany

[2]Univ. Politècnica de Catalunya, Spain

[3]Univ. Karlsruhe, Germany

AofA 2006
Alden Biesen, Belgium

Introduction

- Modern hardware executes several sequential instructions in a pipelined fashion

Introduction

- Modern hardware executes several sequential instructions in a pipelined fashion
- Jump instructions pose a major challenge!

Introduction

- Modern hardware executes several sequential instructions in a pipelined fashion
- Jump instructions pose a major challenge!
- So we try to predict which branch will be taken …

Introduction

- Modern hardware executes several sequential instructions in a pipelined fashion
- Jump instructions pose a major challenge!
- So we try to predict which branch will be taken ...
- Branch mispredictions are expensive: we have to rollback the pipeline

Introduction

- In comparison-based algorithms, we want comparisons to yield as much information as possible $\Longrightarrow$ difficult to predict!

Introduction

- In comparison-based algorithms, we want comparisons to yield as much information as possible $\Longrightarrow$ difficult to predict!
- In static branch prediction, jump instructions are statically predicted as `TAKEN` or `NOT TAKEN`

Introduction

- In comparison-based algorithms, we want comparisons to yield as much information as possible $\Longrightarrow$ difficult to predict!
- In static branch prediction, jump instructions are statically predicted as `TAKEN` or `NOT TAKEN`
- In dynamic branch prediction, the hardware predicts what to do during execution, taking the past into account

Introduction

- In comparison-based algorithms, we want comparisons to yield as much information as possible $\Longrightarrow$ difficult to predict!
- In static branch prediction, jump instructions are statically predicted as `TAKEN` or `NOT TAKEN`
- In dynamic branch prediction, the hardware predicts what to do during execution, taking the past into account
  - 1-bit: We predict the instruction will take the same direction it took the last time it was executed

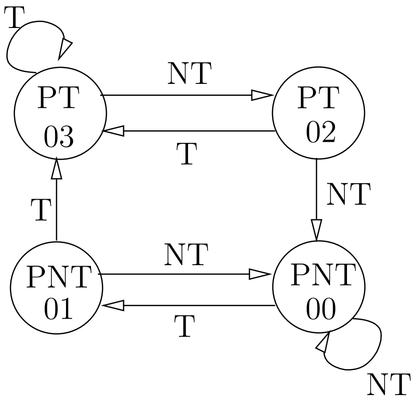
Introduction

- In comparison-based algorithms, we want comparisons to yield as much information as possible $\Longrightarrow$ difficult to predict!
- In static branch prediction, jump instructions are statically predicted as `TAKEN` or `NOT TAKEN`
- In dynamic branch prediction, the hardware predicts what to do during execution, taking the past into account
  - 1-bit: We predict the instruction will take the same direction it took the last time it was executed
  - 2-bit: We must be wrong twice before we change the prediction

Introduction

- In comparison-based algorithms, we want comparisons to yield as much information as possible $\Longrightarrow$ difficult to predict!
- In static branch prediction, jump instructions are statically predicted as TAKEN or NOT TAKEN
- In dynamic branch prediction, the hardware predicts what to do during execution, taking the past into account
  - 1-bit: We predict the instruction will take the same direction it took the last time it was executed
  - 2-bit: We must be wrong twice before we change the prediction
  - ...

2-bit Predictor



Partition

```
    // We have to partition A[i..j] around the pivot
    // that we have already put on A[i]
    int l = i; int u = j + 1; Elem pv = A[i];
    for ( ; ; ) {
       do ++l; while(A[l] < pv);  // Loop S
       do --u; while(A[u] > pv);   // Loop G
       if (l >= u) break;
       swap(A[l], A[u]);
    };
    swap(A[i], A[u]); k = u;
}
```

Setting up the Recurrences

- Probability that the chosen pivot is the kth smallest element out of the n: $\pi_{n,k}$

Setting up the Recurrences

- Probability that the chosen pivot is the kth smallest element out of the n: $\pi_{n,k}$
- Average number of branch mispredictions when partitioning an array of size n and the pivot is the kth: $b_{n,k}$

Setting up the Recurrences

- Probability that the chosen pivot is the kth smallest element out of the n: $\pi_{n,k}$
- Average number of branch mispredictions when partitioning an array of size n and the pivot is the kth: $b_{n,k}$
- Average number of branch mispredictions whan partitioning an array of size n:

$$b_n = \sum_{1 \leq k \leq n} \pi_{n,k} \cdot b_{n,k}$$

Setting up the Recurrences

- Average number of branch mispredictions B_n to sort n elements:

$$B_n = b_n + \sum_{k=1}^{n} \pi_{n,k} \cdot (B_{k-1} + B_{n-k})$$

Setting up the Recurrences

- Average number of branch mispredictions B_n to sort n elements:

$$B_n = b_n + \sum_{k=1}^{n} \pi_{n,k} \cdot (B_{k-1} + B_{n-k})$$

- We will later consider the total cost T_n which satisfies the same recurrence with toll function

$$t_n = n + \xi \cdot b_n + o(n)$$

Sampling

- It is well-known that using samples to select the pivot of each recursive stage improves the average performance of Quicksort and reduces the probability of worst-case behavior

Sampling

- It is well-known that using samples to select the pivot of each recursive stage improves the average performance of quicksort and reduces the probability of worst-case behavior
- For quicksort with samples of size s from which we pick the $(p+1)$th element as the pivot, we have

$$\pi_{n,k} = \frac{\binom{k-1}{p}\binom{n-k}{s-1-p}}{\binom{n}{s}}$$

Sampling

- A typical case is to pick the median of the sample with $s = 2t + 1$ and $p = t$

Sampling

- A typical case is to pick the median of the sample with $s = 2t + 1$ and $p = t$
- We can use variable-size samples with $s = s(n)$; then $s \to \infty$ as $n \to \infty$ but must grow sublinearly, $s = o(n)$; we use ψ to denote the relative rank of the pivot within the sample $\implies$ e.g., $\psi = 1/2$ means choosing the median of the sample

General results

Theorem

The average number of branch mispredictions to sort n elements with quicksort using samples of size s and choosing the $(p+1)$th in the sample of each stage is

$$B_n = \frac{\beta(s,p)}{\mathcal{H}(s,p)} n \ln n + O(n),$$

where

$$\mathcal{H}(s,p) = H_{s+1} - \frac{p+1}{s+1} H_{p+1} - \frac{s-p}{s+1} H_{s-p}.$$

and

$$\beta(s,p) = \lim_{n\to\infty} \frac{b_n}{n} = \lim_{n\to\infty} \frac{1}{n} \sum_{1\le k\le n} \pi_{n,k}^{(s,p)} b_{n,k}$$

General results

Theorem

For variable-sized sampling, if $s \to \infty$ as $n \to \infty$ with $s = o(n)$, and $p/s \to \psi$ then

$$B_n = \frac{\beta(\psi)}{\mathcal{H}(\psi)} n \ln n + o(n \log n),$$

with $\beta(\psi) = \lim_{n\to\infty} \beta(s, \psi \cdot s + o(s))$ and $\mathcal{H}(x) = -(x \ln x + (1-x)\ln(1-x))$

General results

Theorem

The total cost T_n of Quicksort is given by

$$T_n = \frac{1 + \xi \cdot \beta(s,p)}{\mathcal{H}(s,p)} n \ln n + O(n), \qquad s = \Theta(1)$$

and

$$T_n = \frac{1 + \xi \cdot \beta(\psi)}{\mathcal{H}(\psi)} n \ln n + o(n \log n), \qquad s = \omega(1), s = o(n)$$

General results

- In order to compute $\beta(s,p)$, we can use, under suitable conditions,

$$\beta(s,p) = \frac{s!}{p!(s-1-p)!} \int_0^1 x^p (1-x)^{s-1-p} b(x)\, dx$$

with

$$b(x) = \lim_{n\to\infty} \frac{b_{n,x\cdot n}}{n}$$

General results

- In order to compute $\beta(s,p)$, we can use, under suitable conditions,

$$\beta(s,p) = \frac{s!}{p!(s-1-p)!} \int_0^1 x^p (1-x)^{s-1-p} b(x)\, dx$$

  with

$$b(x) = \lim_{n \to \infty} \frac{b_{n,x \cdot n}}{n}$$

- Computing $\beta(\psi)$ is easier!

$$\beta(\psi) = b(\psi)$$

General results

- The optimal value ψ^* for ψ minimizes the total cost, i.e., minimizes

$$\tau_\xi(\psi) = \frac{1 + \xi \cdot \beta(\psi)}{\mathcal{H}(\psi)}$$

and depends on ξ

General results

- The optimal value ψ^* for ψ minimizes the total cost, i.e., minimizes

$$\tau_\xi(\psi) = \frac{1 + \xi \cdot \beta(\psi)}{\mathcal{H}(\psi)}$$

  and depends on ξ
- It's not difficult to prove that for any s and p,

$$\frac{\beta(s,p)}{\mathcal{H}(s,p)} > \frac{\beta(\psi^*)}{\mathcal{H}(\psi^*)}$$

General results

- In general, there exists a threshold value ξ_c such that if $\xi \leq \xi_c$ (branch mispredictions are not too expensive) then we have to take the median of the samples, i.e., $\psi^* = 1/2$

General results

- In general, there exists a threshold value ξ_c such that if $\xi \leq \xi_c$ (branch mispredictions are not too expensive) then we have to take the median of the samples, i.e., $\psi^* = 1/2$
- If $\xi > \xi_c$ (that can happen often in practice!) then $\psi^* < 1/2$ and it is given by the unique solution in $[0, 1/2)$ of the equation

$$\xi \cdot b'(\psi)\mathcal{H}(\psi) = (1 + \xi \cdot b(\psi))\mathcal{H}'(\psi)$$

(provided that $b(x)$ is in $C^2[0, 1/2)$)

General results

- The threshold value ξ_c is the solution of

$$\left.\frac{d^2\tau_\xi(\psi)}{d\psi^2}\right|_{\psi=1/2} = 0$$

General results

- The threshold value ξ_c is the solution of

$$\left.\frac{d^2\tau_\xi(\psi)}{d\psi^2}\right|_{\psi=1/2} = 0$$

- That is

$$\xi_c = -\frac{4}{b''(1/2)\ln 2 + 4b(1/2)}$$

Static Branch prediction

- We analyze here optimal prediction: if the position of the pivot $k \leq n/2$ then we predict Loop S not taken and loop G taken, and the other way around

Static Branch prediction

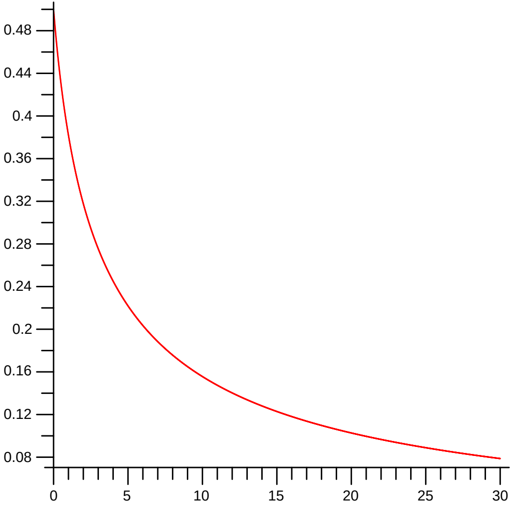
- We analyze here optimal prediction: if the position of the pivot $k \leq n/2$ then we predict Loop S not taken and loop G taken, and the other way around
- If $k \leq n/2$ we incur a branch misprediction every time there is an element which is smaller than the pivot; symetrically, if $k > n/2$ then the number of branch mispredictions is $n - k$

Static branch prediction

- We analyze here optimal prediction: if the position of the pivot $k \leq n/2$ then we predict Loop S not taken and loop G taken, and the other way around
- If $k \leq n/2$ we incur a branch misprediction every time there is an element which is smaller than the pivot; symetrically, if $k > n/2$ then the number of branch mispredictions is $n - k$
- Hence, $b_{n,k} = \min(k-1, n-k)$, $b(\psi) = \min(\psi, 1-\psi)$ and

$$\tau_\xi(\psi) = \frac{1 + \xi \cdot \min(\psi, 1-\psi)}{\mathcal{H}(\psi)}$$

Static Branch prediction



The value of ψ^* as a function of ξ

1-bit branch prediction

- The number of branch mispredictions is twice the number of exchanges: we incur a misprediction each time we abandon the loops S and G

1-bit branch prediction

- The number of branch mispredictions is twice the number of exchanges: we incur a misprediction each time we abandon the loops S and G
- Hence, $b_{n,k} = 2(k-1)(n-k)$ and $b(\psi) = 2\psi(1-\psi)$

1-bit branch prediction

- We can analyze in full detail the performance when using fixed-sized samples. For example, for median-of-$(2t+1)$ we have

$$\beta(2t+1, t) = \frac{t+1}{2t+3}$$

1-Bit Branch prediction

- We can analyze in full detail the performance when using fixed-sized samples. For example, for median-of-$(2t+1)$ we have

$$\beta(2t+1,t) = \frac{t+1}{2t+3}$$

- For variable-size samples, $\beta(\psi) = 2\psi(1-\psi)$.

1-Bit Branch prediction

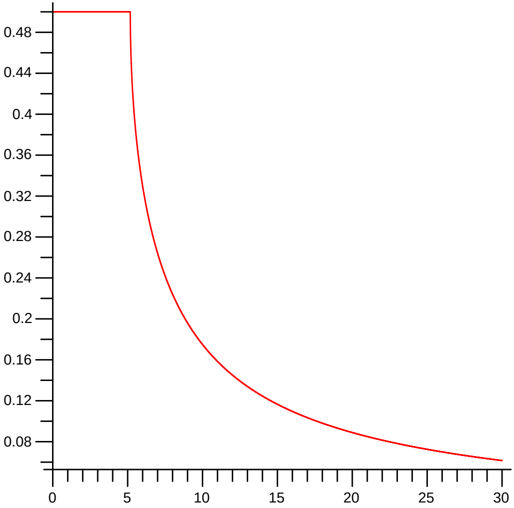
- We can analyze in full detail the performance when using fixed-sized samples. For example, for median-of-$(2t+1)$ we have

$$\beta(2t+1,t) = \frac{t+1}{2t+3}$$

- For variable-size samples, $\beta(\psi) = 2\psi(1-\psi)$.
- The threshold is then at $\xi_c = 2/(2\ln 2 - 1) \approx 5.177\ldots$ and ψ^* is the solution of

$$\ln\psi + 2\xi\psi^2\ln\psi = \ln(1-\psi) + 2\xi(1-\psi)^2\ln(1-\psi)$$

1-Bit Branch prediction



The value of ψ^* as a function of ξ

2-Bit Branch prediction

- In (Kaligosi, Sanders, 2006), an approximate model to compute $b_{n,k}$ is given, from which

$$b(x) = \frac{2x^4 - 4x^3 + x^2 + x}{1 - x(1 - x)}$$

follows

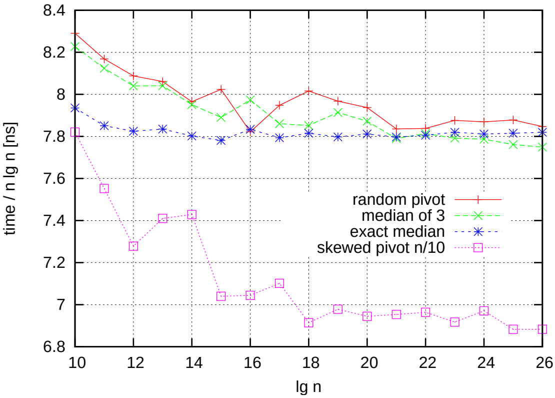
2-Bit Branch prediction

- In (Kaligosi, Sanders, 2006), an approximate model to compute $b_{n,k}$ is given, from which

$$b(x) = \frac{2x^4 - 4x^3 + x^2 + x}{1 - x(1 - x)}$$

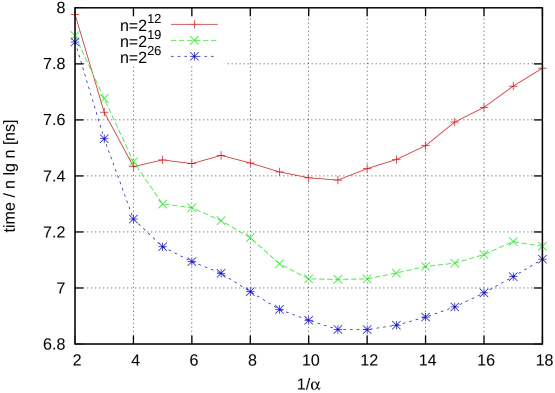
  follows
- We are working on a more refined analysis of $b_{n,k}$ for this prediction scheme; once $b_{n,k}$ has been found, we should only have to apply the machinery shown here

Some real data



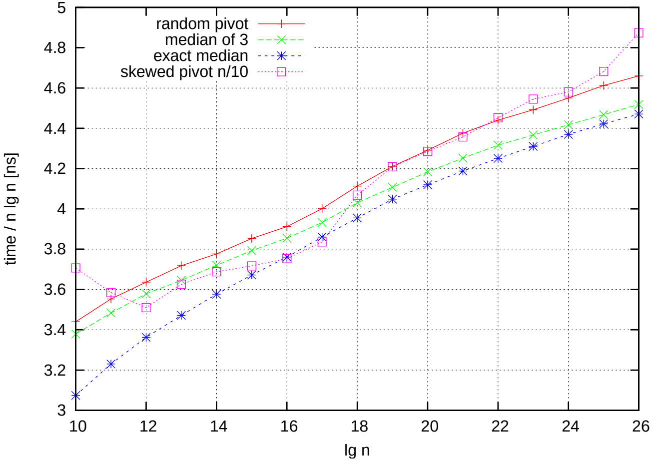
Time vs. size on a Pentium 4 (from (Kaligosi, Sanders, 2006))

Some real data



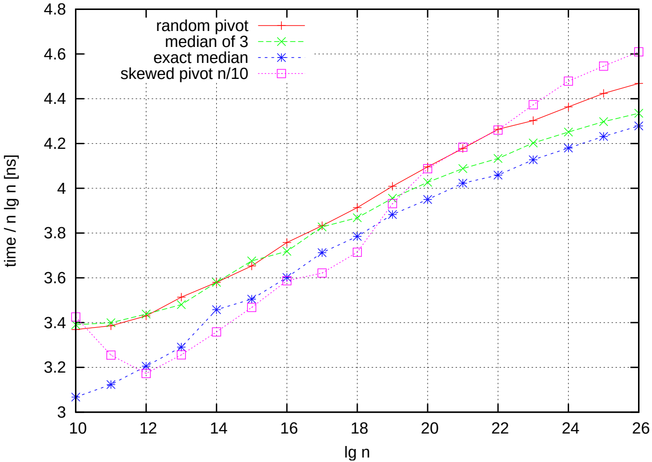
Time vs. $1/\psi$ on a Pentium 4

Some real data



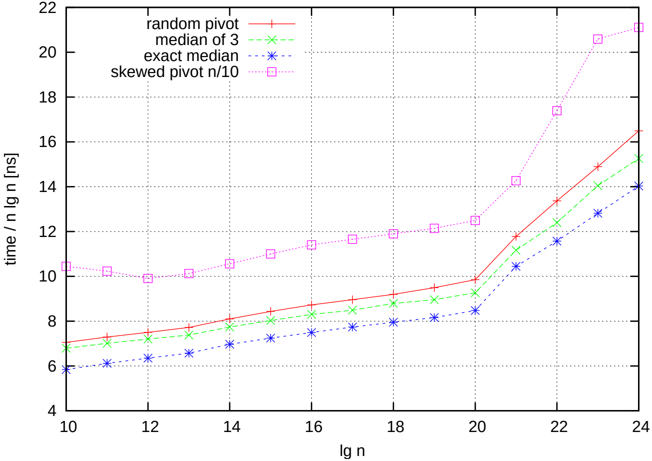
Time vs. size on an Athlon 64

Some real data



Time vs. size on an Opteron

Some real data



Time vs. size on a Sun

Future work

- Complete the analysis of static branch prediction with fixed-size samples (it's not easy to obtain $\beta(s, p)$ for general s and p!)
- Analyze the 2-bit prediction scheme and possibly others
- Conduct additional experiments, compare theoretical analysis to real data
- Analyze branch mispredictions and their impact on the performance of other algorithms