SIMD instructions in Hashtables

Pablo Rotondo

LIGM, Université Gustave Eiffel

Joint work with

Cyril Nicaud (LIGM)

Séminaire LIGM,

Champs-Sur-Marne, 27 January, 2026.

Introduction

- Aim: study and model actual implementations
  - Engineers sometimes choose innovative implementations

    e.g., `TimSort` in `Python`.
  - Study choices in depth, make recommendations.

Introduction

- Aim: study and model actual implementations
  - Engineers sometimes choose innovative implementations
    e.g., `TimSort` in `Python`.
  - Study choices in depth, make recommendations.
- Why not use textbook solutions?
  - typical usage of data structures,
  - architectural features of modern computers.

Introduction

- Aim: study and model actual implementations
  - Engineers sometimes choose innovative implementations
    e.g., `TimSort` in `Python`.
  - Study choices in depth, make recommendations.
- Why not use textbook solutions?
  - typical usage of data structures,
  - architectural features of modern computers.

⇒ This talk: SIMD instructions (Single Instruction, Multiple Data) in modern HashTables.

A crash course in HashTables

Motivation

Implement an associative array m:

- universe $\mathcal{U}$ of keys $k \in \mathcal{U}$ enormous,
- associate some keys k to values $m[k]$,
- insert, search, delete...

A crash course in HashTables

Motivation

Implement an associative array m:

- universe $\mathcal{U}$ of keys $k \in \mathcal{U}$ enormous,
- associate some keys k to values $m[k]$,
- insert, search, delete...

Hashtables:

- **Idea** : use a small array A, of size $n \ll |\mathcal{U}|$
  - consider $h : \mathcal{U} \to \mathbb{Z}$ *pseudo-random*,
  - insert k in bucket $A[i]$ where $i = h(k) \bmod n$.
- **Problem** : collisions, *keys* k_1 and k_2 with $h(k_1) = h(k_2)$.

A crash course in HashTables 2

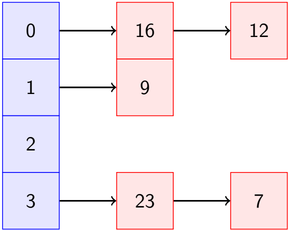
- **Collision resolution policy** :

  1. [External Hashing / Closed addressing] Each bucket $A[i]$ contains a linked list.
  2. [Internal Hashing / Open addressing] If bucket is occupied, find another that is free.

A crash course in HashTables 2

- **Collision resolution policy** :

  1. [External Hashing / Closed addressing] Each bucket $A[i]$ contains a linked list.



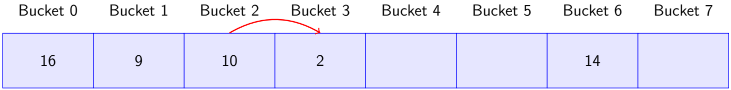
  2. [Internal Hashing / Open addressing] If bucket is occupied, find another that is free.

A crash course in HashTables 2

- **Collision resolution policy** :

  1. [External Hashing / Closed addressing] Each bucket $A[i]$ contains a linked list.

  2. [Internal Hashing / Open addressing] If bucket is occupied, find another that is free.



[red arrow is indicative]

A crash course in HashTables 2

- **Collision resolution policy** :

  1. [External Hashing / Closed addressing] Each bucket $A[i]$ contains a linked list.

  2. [Internal Hashing / Open addressing] If bucket is occupied, find another that is free.

- **Rehashing policy:**

  – if *load factor* $\frac{\#keys}{n}$ is "big" $(> \theta \in (0,1))$, create larger array,
  – must reinsert everything. [slow! but amortized overall]

Internal hashing / Open addressing

- Internal hashing / Open addressing more popular when looking for performance (better locality).
- Many strategies to define the *probe sequence*.

Internal hashing / Open addressing

- Internal hashing / Open addressing more popular when looking for performance (better locality).
- Many strategies to define the *probe sequence*.

Probing sequence

To search/insert x:

- Start at $i_0 = h(x) \bmod n$.
- If occupied, test $i_1, i_2, \ldots$ etc. in order.

Internal hashing / Open addressing

- Internal hashing / Open addressing more popular when looking for performance (better locality).
- Many strategies to define the *probe sequence*.

Probing sequence

To search/insert x:

- Start at $i_0 = h(x) \bmod n$.
- If occupied, test $i_1, i_2, \ldots$ etc. in order.

Modulo n,

- **Linear probing:** $i_1 = i_0 + 1$, $i_2 = i_1 + 1$, $\ldots$
- **Quadratic probing:** $i_1 = i_0 + 1$, $i_2 = i_1 + 2$, $\ldots$, $i_j = i_{j-1} + j, \ldots$
- **Double hashing:** $\Delta(x) = h_2(x)$, $i_1 = i_0 + \Delta, i_2 = i_1 + \Delta, \ldots$

Internal hashing / Open addressing

- Internal hashing / Open addressing more popular when looking for performance (better locality).
- Many strategies to define the *probe sequence*.

Probing sequence

To search/insert x:

- Start at $i_0 = h(x) \bmod n$.
- If occupied, test $i_1, i_2, \ldots$ etc. in order.

Modulo n,

- **Linear probing:** $i_1 = i_0 + 1$, $i_2 = i_1 + 1$, $\ldots$
- **Quadratic probing:** $i_1 = i_0 + 1$, $i_2 = i_1 + 2$, $\ldots$, $i_j = i_{j-1} + j, \ldots$
- **Double hashing:** $\Delta(x) = h_2(x)$, $i_1 = i_0 + \Delta, i_2 = i_1 + \Delta, \ldots$

Theoretical model: random probing model, each i_j is taken uniformly and independently from $\{0, \ldots, n-1\}$.

Using vectorization in hash tables

SIMD in a nutshell: large registers seen as vectors of several lanes.

Using vectorization in hash tables

SIMD in a nutshell: large registers seen as vectors of several lanes.

Example: comparing vectors of 128 bits with 16 lanes of 1 byte

4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

9	5	-2	4	7	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

↓ `_mm_cmpeq_epi8`

0	0	0	-1	0	-1	0	0	0	0	0	0	0	0	0	0

Using vectorization in hash tables

SIMD in a nutshell: large registers seen as vectors of several lanes.

Example: comparing vectors of 128 bits with 16 lanes of 1 byte

4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

9	5	-2	4	7	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

↓ _mm_cmpeq_epi8

0	0	0	-1	0	-1	0	0	0	0	0	0	0	0	0	0

Idea: buckets of b keys, keep header of metadata for each bucket

- reduced hash value (usually one byte) for each key
- when searching/inserting, compare reduced hash first

Using vectorization in hash tables

SIMD in a nutshell: large registers seen as vectors of several lanes.

Example: comparing vectors of 128 bits with 16 lanes of 1 byte

4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

9	5	-2	4	7	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

↓ `_mm_cmpeq_epi8`

0	0	0	-1	0	-1	0	0	0	0	0	0	0	0	0	0

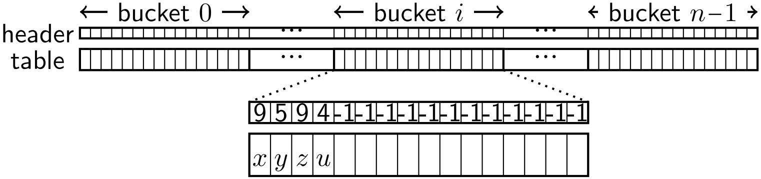
Idea: buckets of b keys, keep header of metadata for each bucket

- reduced hash value (usually one byte) for each key
- when searching/inserting, compare reduced hash first

Principle of several modern hashtables:

`boost::unordered_flat_map` ($b = 15$), `Google Abseil Swiss tables` ($b = 16$), `F14` of Meta ($b = 14$)...

Using vectorization in hash tables 2



We consider the *fingerprint byte*

- to be $255 \equiv -1$ when the position is free,
- to be $254 \equiv -2$ when the position was deleted, [tombstone]
- else we have a reduced *hash value*.

Our model: parameters

We are interested in *hops* : [accesses to buckets]

- number of buckets accessed for a successful search,
- proportion of full buckets.

Our model: parameters

We are interested in *hops* : [accesses to buckets]

- number of buckets accessed for a successful search,
- proportion of full buckets.

Insert k distinct elements into a table with n buckets:

- probe sequence made of random numbers $\{0, \ldots, n-1\}$.
- for the moment we do not consider deletions,

Later in talk: unsuccessful search.

Main results: proportion of occupancy

We are interested in *hops* :

- number of buckets accessed for a successful search,
- **proportion of full buckets.**

Main results: proportion of occupancy

We are interested in *hops* :

- number of buckets accessed for a successful search,
- **proportion of full buckets.**

Let $U_j(k)$: # buckets with j occupied slots after k insertions,

Theorem [Nicaud, R, 26+]: proportion of occupancy

Suppose $k \leq \theta \cdot N$ for $\theta \in (0,1)$, $N := b \times n$ the maximum capacity.

With probability tending to one, we have uniformly

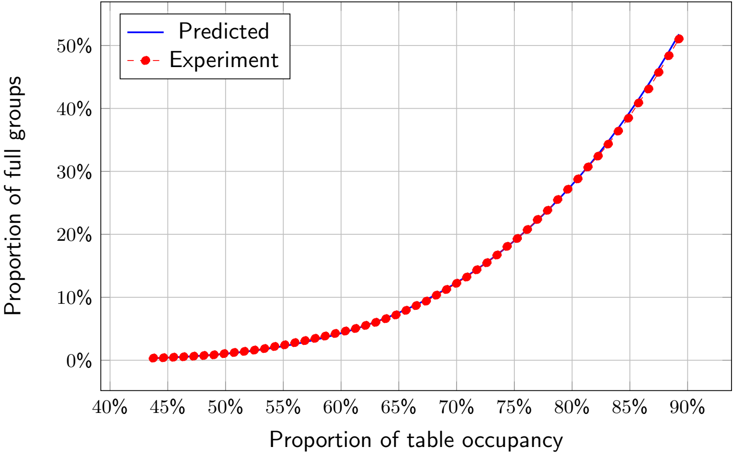
$$U_j(k) = n u_j(k/N) + o(n) \,, \qquad ,$$

where $u_j(t) = \frac{\lambda(t)^j}{j!} e^{-\lambda(t)}$ for $j = 0, \ldots, b-1$, $u_b(t) = 1 - \sum_{j<b} u_j(t)$ and $\lambda(t) = \lambda_b(t)$ is a special function we can compute[a].

[a] Note that $S(k) := \sum j U_j(k)$ increases always by 1, so $\sum j u_j(k) = bt$.

Proportion of occupancy

Plot of a *single* run with $n = 2^{14} = 16\ 384$, $b = 15$,



Stopped at $\theta = 0.875$, maximum load of `boost::unordered_flat_map`.

Main results: number of hops

The special function $\lambda(t)$ is associated to hops.

Let $R(k)$: be the total number of hops up the the k-th insertion,

Main results: number of hops

The special function $\lambda(t)$ is associated to hops.

Let $R(k)$: be the total number of hops up the the k-th insertion,

Theorem [Nicaud, R, 26+]: number of hops

Suppose $k \leq \theta \cdot N$ for $\theta \in (0,1)$, $N := b \times n$ the maximum capacity. With probability tending to one,

$$R(k) = n\lambda_b(k/N) + o(n)\,,$$

$\lambda_b(t)$ is defined implicitly by

$$e^{-\lambda_b(t)} \sum_{i<b} (b-i)\frac{1}{i!}\lambda_b(t)^i = b - bt\,.$$

Successful search

We consider the usual model for **successful search**:

- Pick an element uniformly at random from the k present.
- Classical model, others are possible.

Successful search

We consider the usual model for **successful search**:

- Pick an element uniformly at random from the k present.
- Classical model, others are possible.

Corollary (Successful search)

With high probability, the average successful search time is $\lambda_b(t)/(bt) + o(1)$, *where* $t = k/N$.

Successful search

We consider the usual model for **successful search**:

- Pick an element uniformly at random from the k present.
- Classical model, others are possible.

Corollary (Successful search)

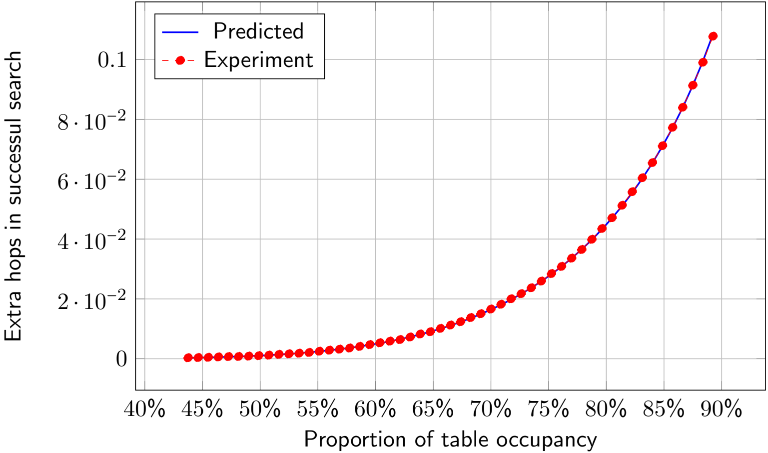
With high probability, the average successful search time is $\lambda_b(t)/(bt) + o(1)$, where $t = k/N$.

Proof.

The successful search time is $R(k)/k$. □

Successful search

Plot of a *single* run with $n = 2^{14} = 16\ 384$, $b = 15$,



Stopped at $\theta = 0.875$, maximum load of `boost::unordered_flat_map`.

The hops function $\lambda_b(t)$

- The case $b = 1$: $\lambda_1(t) = \log(\frac{1}{1-t})$. Successful search time coincides with the classical $\frac{1}{t}\log(\frac{1}{1-t})$.
- The case $b = 2$: using the -1-th branch of the Lambert-W

$$\lambda_b(t) = -2 - W_{-1}\left(-\frac{2-2t}{e^2}\right) .$$

The hops function $\lambda_b(t)$

- The case $b = 1$: $\lambda_1(t) = \log(\frac{1}{1-t})$. Successful search time coincides with the classical $\frac{1}{t}\log(\frac{1}{1-t})$.
- The case $b = 2$: using the -1-th branch of the Lambert-W

$$\lambda_b(t) = -2 - W_{-1}\left(-\frac{2-2t}{e^2}\right).$$

Still, it remains logarithmic in order

Proposition

We have $\log\left(\frac{1}{1-t}\right) \le \lambda_b(t) \le b\log\left(\frac{1}{1-t}\right)$ for all t. Moreover $\lambda_b(t) \sim \log\left(\frac{1}{1-t}\right)$ as $t \to 1^-$ and $\lambda_b(t) \sim b\log\left(\frac{1}{1-t}\right)$ as $t \to 0^+$.

The hops function $\lambda_b(t)$

- The case $b = 1$: $\lambda_1(t) = \log(\frac{1}{1-t})$. Successful search time coincides with the classical $\frac{1}{t}\log(\frac{1}{1-t})$.
- The case $b = 2$: using the -1-th branch of the Lambert-W

$$\lambda_b(t) = -2 - W_{-1}\left(-\frac{2-2t}{e^2}\right).$$

Still, it remains logarithmic in order

Proposition

We have $\log\left(\frac{1}{1-t}\right) \le \lambda_b(t) \le b\log\left(\frac{1}{1-t}\right)$ for all t. Moreover $\lambda_b(t) \sim \log\left(\frac{1}{1-t}\right)$ as $t \to 1^-$ and $\lambda_b(t) \sim b\log\left(\frac{1}{1-t}\right)$ as $t \to 0^+$.

- Cost of a successful search is related to $\lambda_b(t)/b$.
- Effect is not simply dividing the search time by b !

Plan of the talk

1. Introduction
2. Model and first results: successful search
3. Elements of the proof: techniques
4. Model and result for unsuccessful search
5. Conclusions and further work

A probabilistic recurrence: simulating occupancy

Let $U_j(k)$: # buckets with j occupied slots after k insertions.

Given the situation at time k, which we call $\mathcal{F}_k$,

$$U_j(k+1) - U_j(k) = \begin{cases} +1, & \text{with probability } \frac{U_{j-1}(k)}{n - U_b(k)} \text{ if } j > 0\,, \\ -1, & \text{with probability } \frac{U_j(k)}{n - U_b(k)} \text{ if } j < b\,, \\ 0, & \text{otherwise.} \end{cases}$$

A probabilistic recurrence: simulating occupancy

Let $U_j(k)$: # buckets with j occupied slots after k insertions.

Given the situation at time k, which we call $\mathcal{F}_k$,

$$U_j(k+1) - U_j(k) = \begin{cases} +1, & \text{with probability } \frac{U_{j-1}(k)}{n-U_b(k)} \text{ if } j > 0\,, \\ -1, & \text{with probability } \frac{U_j(k)}{n-U_b(k)} \text{ if } j < b\,, \\ 0, & \text{otherwise.} \end{cases}$$

We have the conditional expectation

$$\mathbb{E}[U_j(k+1) - U_j(k) \mid \mathcal{F}_k] = \mathbf{1}_{j>0} \frac{\frac{U_{j-1}(k)}{n}}{1 - \frac{U_b(k)}{n}} - \mathbf{1}_{j<b} \frac{\frac{U_j(k)}{n}}{1 - \frac{U_b(k)}{n}}\,.$$

Wormald's Differential Equation Method

- Method introduced to study dynamics on graphs.

Wormald's Differential Equation Method

- Method introduced to study dynamics on graphs.
- System of equations for conditional expectations

$$\mathbb{E}[Y_j(k+1) - Y_j(k) \mid \mathcal{F}_k] = f_j(k/N; Y_1(k)/N, \ldots, Y_\ell(k)/N),$$

$j = 1, \ldots, \ell.$

Wormald's Differential Equation Method

- Method introduced to study dynamics on graphs.
- System of equations for conditional expectations

  $$\mathbb{E}\left[Y_j(k+1) - Y_j(k) \;\middle|\; \mathcal{F}_k\right] = f_j(k/N; Y_1(k)/N, \ldots, Y_\ell(k)/N)\,,$$

  $j = 1, \ldots, \ell.$
- If $|Y_j(k+1) - Y_j(k)|$ is "small" and functions f_j are "regular":

  $$Y_j(k) \approx N y_j(k/N)\,,$$

  with probability tending to one, where

  $$\begin{cases} y_1'(t) & = f_1(t; y_1(t), \ldots, y_\ell(t))\,, \\ & \vdots \\ y_\ell'(t) & = f_\ell(t; y_1(t), \ldots, y_\ell(t))\,. \end{cases}$$

Wormald's Differential Equation Method 2

Theorem (version adapted from Warnke'19)

Consider $\beta = \beta(N)$, $\gamma := \gamma(N) \to 0$, $\lambda := \lambda(N) \to 0$,

1. *for all* $k < N$ *and* $j \in [\ell]$,
   $\Pr(|Y_j(k+1) - Y_j(k)| \leq \beta(N)) \geq 1 - \gamma(N)$;
2. *for all* $j \in [\ell]$, *and* $k < N$,
   $\mathbb{E}[Y_j(k+1) - Y_j(k) \mid \mathcal{F}_k] = f_j(k/N; Y_1(k)/N, \ldots, Y_\ell(k)/N)$,
   where the functions f_j *are* L*-Lipschitz on a domain* $\mathcal{D}$.
3. *The initial* $\boldsymbol{Y}(0) = (Y_1(0), \ldots, Y_\ell(0))$ *satisfies*
   $\|\boldsymbol{Y}(0) - N\boldsymbol{v}\|_\infty \leq \lambda \cdot N$ *for* $(0, v_1, \ldots, v_\ell) \in \mathcal{D}$.

– *Let* $\boldsymbol{y}(t) = (y_1(t), \ldots, y_\ell(t))$ *be the solution to the system of differential equations with* $y_1(0) = v_1, \ldots, y_\ell(0) = v_\ell$.
– *Let* $\theta > 0$ *be such that* $(s, \boldsymbol{y}(s)) \in \mathcal{D}$ *for all* $s \in [0, \theta]$.

For $\lambda(N) = \Omega(1/N)$, *with proba* $\geq 1 - \theta\ell N\gamma - 2b\exp(-N\lambda^2/(8\theta\beta^2))$,

$$|Y_j(k) - Ny_j(k/N)| \leq 3e^{L\theta}\lambda(N) \cdot N = o(N),$$

for all $j \in [\ell]$ *and* $k \leq \theta N$.

Wormald's Differential Equation Method 3

Conclusion of the Theorem

For $\lambda(N) = \Omega(1/N)$, with proba $\geq 1 - \theta\ell N\gamma - 2b\exp(-N\lambda^2/(8\theta\beta^2))$,

$$|Y_j(k) - Ny_j(k/N)| \leq 3e^{L\theta}\lambda(N) \cdot N = o(N)$$

for all $j \in [\ell]$ and $k \leq \theta N$.

To apply the result, we require

- $\beta(N)$ can tend to infinity, but not too fast,
- $\gamma(N)$ should tend to 0 with $N\gamma(N) \to 0$,
- $\lambda(N) \to 0$ but not too fast.

Wormald's Differential Equation Method 3

Conclusion of the Theorem

For $\lambda(N) = \Omega(1/N)$, with proba $\geq 1 - \theta\ell N\gamma - 2b\exp(-N\lambda^2/(8\theta\beta^2))$,

$$|Y_j(k) - Ny_j(k/N)| \leq 3e^{L\theta}\lambda(N) \cdot N = o(N)$$

for all $j \in [\ell]$ and $k \leq \theta N$.

To apply the result, we require

- $\beta(N)$ can tend to infinity, but not too fast,
- $\gamma(N)$ should tend to 0 with $N\gamma(N) \to 0$,
- $\lambda(N) \to 0$ but not too fast.

– For the case of $U_j(k)$ we simply have $\beta(N) = 1$ and $\lambda(N)$ can be any function such that $N\lambda^2(N) \to \infty$.

Wormald's Differential Equation Method 3

Conclusion of the Theorem

For $\lambda(N) = \Omega(1/N)$, with proba $\geq 1 - \theta\ell N\gamma - 2b\exp(-N\lambda^2/(8\theta\beta^2))$,

$$|Y_j(k) - Ny_j(k/N)| \leq 3e^{L\theta}\lambda(N) \cdot N = o(N)$$

for all $j \in [\ell]$ and $k \leq \theta N$.

To apply the result, we require

- $\beta(N)$ can tend to infinity, but not too fast,
- $\gamma(N)$ should tend to 0 with $N\gamma(N) \to 0$,
- $\lambda(N) \to 0$ but not too fast.

– For the case of $U_j(k)$ we simply have $\beta(N) = 1$ and $\lambda(N)$ can be any function such that $N\lambda^2(N) \to \infty$.

– For the case of $R(k)$, we can prove $R(k+1) - R(k) > (\log N)^2$ with proba $O(e^{-(1-\theta)\times(\log N)^2})$.

Wormald's Differential Equation Method 4

– In the case of $U_j(k)$ we obtain the system:

$$\begin{cases} u_0'(t) & = -b\frac{u_0(t)}{1-u_b(t)}\,, \\ u_i'(t) & = b\frac{u_{i-1}(t)}{1-u_b(t)} - b\frac{u_i(t)}{1-u_b(t)}\,,\ \text{for } i \in \{1,\ldots,b-1\}, \\ u_b'(t) & = b\frac{u_{b-1}(t)}{1-u_b(t)}\,. \end{cases}$$

Wormald's Differential Equation Method 4

– In the case of $U_j(k)$ we obtain the system:

$$\begin{cases} u_0'(t) & = -b\frac{u_0(t)}{1-u_b(t)}\,, \\ u_i'(t) & = b\frac{u_{i-1}(t)}{1-u_b(t)} - b\frac{u_i(t)}{1-u_b(t)}\,,\ \text{for } i \in \{1,\ldots,b-1\}, \\ u_b'(t) & = b\frac{u_{b-1}(t)}{1-u_b(t)}\,. \end{cases}$$

This system appears, from a different process in the work of Wormald.

Wormald's Differential Equation Method 4

– In the case of $U_j(k)$ we obtain the system:

$$\begin{cases} u_0'(t) & = -b\frac{u_0(t)}{1-u_b(t)}\,, \\ u_i'(t) & = b\frac{u_{i-1}(t)}{1-u_b(t)} - b\frac{u_i(t)}{1-u_b(t)}\,,\ \text{for } i \in \{1,\dots,b-1\}, \\ u_b'(t) & = b\frac{u_{b-1}(t)}{1-u_b(t)}\,. \end{cases}$$

This system appears, from a different process in the work of Wormald.

– For $R(k)$ the expected value is

$$\mathbb{E}[R(k+1) - R(k) \big| \mathcal{F}_k] = \sum_{r\geq 1} r(1-U_b(k)/n)(U_b(k)/n)^{r-1} = \frac{1}{1 - {U_b(k)}\big/{n}}$$

which relates to $\lambda_b(t)$ through $\lambda_b(t) = b\int_0^t \frac{dt}{1-u_b(t)}$.

Intuition behind the Differential Equation Method

Why such a strong concentration?[1]

- Underlying martingale:
  $Z_i(k) := \sum_{a<k}(Y_i(a+1) - Y_i(a) - f_i(Y_1(a)/N, \ldots, Y_\ell(a)/N))$,
  with $\mathbb{E}[Z_i(k)] = 0$.
- Maximal Azuma-Hoeffding Theorem implies $Z_i(k) \approx 0$ with high probability.
- Convenient rewriting

  $$Y_i(k) - Ny_i(k/N) = Z_i(k) + \sum_{a<k}\left(f_i\left(\frac{Y_1(a)}{N}, \ldots, \frac{Y_\ell(a)}{N}\right) - N\left(y_i\left(\frac{a+1}{N}\right) - y_i\left(\frac{a}{N}\right)\right)\right),$$

  helps prove bound high probability bound by recurrence.

[1] This is the proof from Warnke'19.

Model for unsuccessful search

Insert k distinct elements into a table with n buckets:

- we search for a key not in the table,
- probe sequence made of random numbers $\{0, \ldots, n-1\}$,

Model for unsuccessful search

Insert k distinct elements into a table with n buckets:

- we search for a key not in the table,
- probe sequence made of random numbers $\{0, \ldots, n-1\}$,

When to stop?

- as there are no deletions, we may stop if bucket is not full,
- `boost::unordered_flat_map` uses *overflow bits*.

Model for unsuccessful search

Insert k distinct elements into a table with n buckets:

- we search for a key not in the table,
- probe sequence made of random numbers $\{0, \ldots, n-1\}$,

When to stop?

- as there are no deletions, we may stop if bucket is not full,
- `boost::unordered_flat_map` uses *overflow bits*.

Overflow bits

- when no space to insert key, leave a “continue mark”,
- use d bits, the overflow bit is a *hash value* in $\{0, \ldots, d-1\}$,
- this allows us to stop a search prematurely,
- more *meta-data* per bucket.

Model for unsuccessful search

Insert k distinct elements into a table with n buckets:

- we search for a key not in the table,
- probe sequence made of random numbers $\{0, \ldots, n-1\}$,

When to stop?

- as there are no deletions, we may stop if bucket is not full,
- `boost::unordered_flat_map` uses *overflow bits*.

Overflow bits

- when no space to insert key, leave a "continue mark",
- use d bits, the overflow bit is a *hash value* in $\{0, \ldots, d-1\}$,
- this allows us to stop a search prematurely,
- more *meta-data* per bucket.

$\Rightarrow$ During insertion, we consider a fresh uniform and independent overflow bit each time needed.

Model for unsuccessful search

In our model

- each overflow bit is a fresh random number in $\{0, \dots, d-1\}$,
- if several are needed, they are independent.

Model for unsuccessful search

In our model

- each overflow bit is a fresh random number in $\{0, \ldots, d-1\}$,
- if several are needed, they are independent.

Let $V_j(k)$ be the number of full buckets with j overflow bits on:

- Number of hops in unsuccessful search is geometric, parameter

$$p = \sum_{j=0}^{d} \frac{V_j(k)}{n} \times \frac{j}{d}\,.$$

- The expected number of hops is $\frac{1}{1-p}$.

Model for unsuccessful search

In our model

- each overflow bit is a fresh random number in $\{0, \ldots, d-1\}$,
- if several are needed, they are independent.

Let $V_j(k)$ be the number of full buckets with j overflow bits on:

- Number of hops in unsuccessful search is geometric, parameter

$$p = \sum_{j=0}^{d} \frac{V_j(k)}{n} \times \frac{j}{d}\,.$$

- The expected number of hops is $\frac{1}{1-p}$.

We study $V_j(k)$ for $j = 0, \ldots, d$. Observe $\sum_j V_j(k) = U_b(k)$.

Main result for unsuccessful search

Theorem (Nicaud, R., 26+)

Starting from an empty table with n buckets of size b, and d overflow bits, the number of full buckets with j overflow bits on at time k, $V_j(k)$, satisfies[a] $V_j(k) = nv_j(k/N) + o(n)$ *with probability tending to one, for all $k \leq \theta N$, where*

$$v_j(t) = \binom{d}{j} e^{-(d-j)\lambda_b(t)/d} \int_0^{\lambda_b(t)} \tfrac{x^{b-1}}{(b-1)!} (e^{-x/d} - e^{-\lambda_b(t)/d})^j \, dx \, .$$

[a] o term is uniform in k and j.

Main result for unsuccessful search

Theorem (Nicaud, R., 26+)

Starting from an empty table with n buckets of size b, and d overflow bits, the number of full buckets with j overflow bits on at time k, $V_j(k)$, satisfies[a] $V_j(k) = nv_j(k/N) + o(n)$ *with probability tending to one, for all $k \le \theta N$, where*

$$v_j(t) = \binom{d}{j} e^{-(d-j)\lambda_b(t)/d} \int_0^{\lambda_b(t)} \tfrac{x^{b-1}}{(b-1)!} (e^{-x/d} - e^{-\lambda_b(t)/d})^j \, dx \, .$$

[a] o term is uniform in k and j.

Corollary

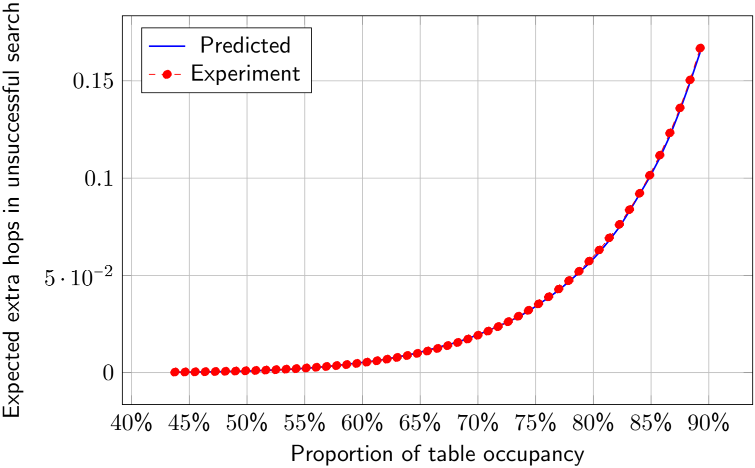
The expected cost of an unsuccessful search is

$$\left(1 - \int_0^{\lambda_b(k/N)} \tfrac{x^{b-1}}{(b-1)!} e^{-(1-1/d)x} (e^{-x/d} - e^{-\lambda_b(k/N)/d}) dx\right)^{-1} + o(1) \, ,$$

with probability tending to one.

Expected cost of unsuccessful search

Plot of a *single* run with $n = 2^{14} = 16\ 384$, $b = 15$, $d = 8$



The case $b = 1$

Overflow bits can be used even when buckets are of size $b = 1$:

- The idea for $b = d = 1$ appears in *Amble and Knuth. Ordered hash tables.* from 1974.
- When $b = 1$ we have $\lambda_b(t) = \log(\frac{1}{1-t})$.
- The unsuccessful search times are

$$\frac{1}{1-t} \times \frac{1}{1 + \frac{(1-t)^{-(1-1/d)}-1}{1-1/d}}\,, \quad (d > 1)\,, \qquad \frac{1}{1-t} \times \frac{1}{1 + \log(\frac{1}{1-t})}\,, \quad (d = 1)\,,$$

compared to $\frac{1}{1-t}$ without overflow bits.

Elements of the study of $V_j(k)$

– Change the notion of time:
  - instead of k insertions consider r hops,
  - k insertions corresponds to $r = R(k)$ hops.

Elements of the study of $V_j(k)$

– Change the notion of time:

- instead of k insertions consider r hops,
- k insertions corresponds to $r = R(k)$ hops.

– In the time-scale of hops

$$\mathbb{E}[\tilde{V}_j(r+1) - \tilde{V}_j(r) \,|\, \tilde{\mathcal{F}}_r] = (1 - \tfrac{j-1}{d})\tilde{V}_{j-1}(r) - (1 - \tfrac{j}{d})\tilde{V}_j(r)\,,$$

for $j > 0$, and $\mathbb{E}[\tilde{V}_0(r+1) - \tilde{V}_0(r) \,|\, \tilde{\mathcal{F}}_r] = \tilde{U}_{b-1}(r) - \tilde{V}_0(r)$.

Elements of the study of $V_j(k)$

– Change the notion of time:

- instead of k insertions consider r hops,
- k insertions corresponds to $r = R(k)$ hops.

– In the time-scale of hops

$$\mathbb{E}[\tilde{V}_j(r+1) - \tilde{V}_j(r) \,\big|\, \tilde{\mathcal{F}}_r] = (1 - \tfrac{j-1}{d})\tilde{V}_{j-1}(r) - (1 - \tfrac{j}{d})\tilde{V}_j(r)\,,$$

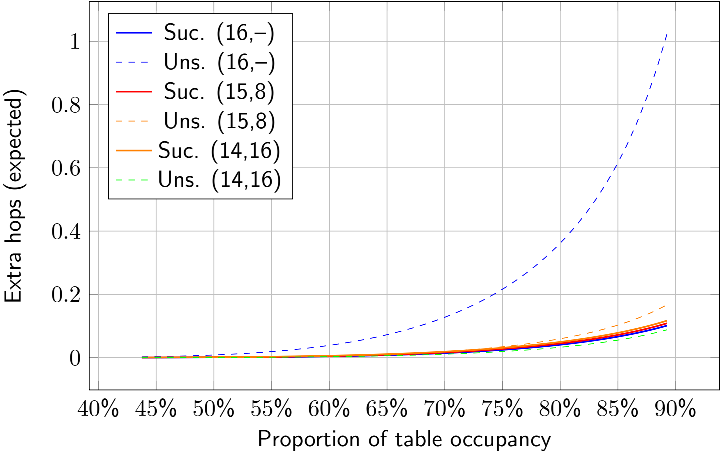
for $j > 0$, and $\mathbb{E}[\tilde{V}_0(r+1) - \tilde{V}_0(r) \,\big|\, \tilde{\mathcal{F}}_r] = \tilde{U}_{b-1}(r) - \tilde{V}_0(r)$.

– We use the Differential Equation Method, and then

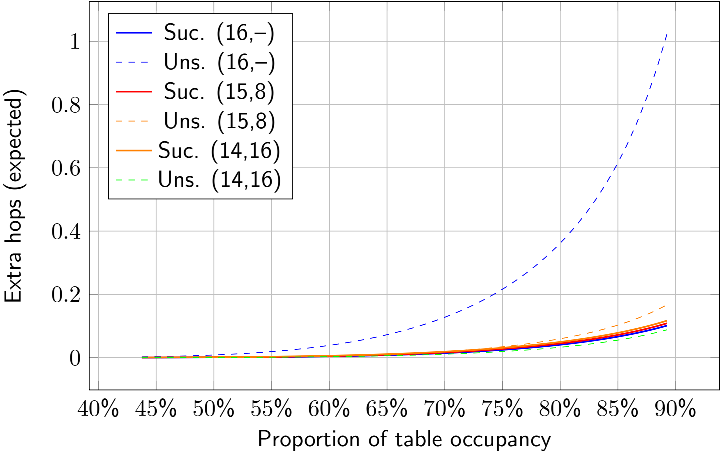
$$R(k) = \tfrac{N}{b}\lambda_b(k/N) + o(n)\,,$$

to link both time-scales as $V_j(k) = \tilde{V}_j(R(k))$.

Different combinations of b and d



Different combinations of b and d



How to make sense of the difference ?

- number of bytes per bucket is the same in all three,
- here we talk about % of full, but capacities are different !
- compare when the number of inserted elements is the same ?

Recap and conclusions

- The current implementation in *boost* is more complex
  buckets are not separated, they overlap.
- Other models of pass-bits exist
  - same bit used every time we leave a mark[2],
  - a counter of "passing" keys instead [F14 of Meta].
- Model with suppression?

[2] P M Martini and W A Burkhard. Double hashing with multiple passbits. IJCS, 14(06):1165–1182, 2003.

Recap and conclusions

- The current implementation in *boost* is more complex
  buckets are not separated, they overlap.
- Other models of pass-bits exist
  - same bit used every time we leave a mark[2],
  - a counter of “passing” keys instead [`F14` of Meta].
- Model with suppression? $\Rightarrow U_{i,j}(k) = \#\ i$ elements and j on bits.

[2] P M Martini and W A Burkhard. Double hashing with multiple passbits. IJCS, 14(06):1165–1182, 2003.

Recap and conclusions

- The current implementation in *boost* is more complex
  buckets are not separated, they overlap.
- Other models of pass-bits exist
  - same bit used every time we leave a mark[2],
  - a counter of "passing" keys instead [`F14` of Meta].
- Model with suppression? $\Rightarrow U_{i,j}(k) = \#\ i$ elements and j on bits.

Conclusions

- We have shown how techniques from graphs dynamics can be applied to hash tables.
- Results are very precise and hold with high probability.
- Introduction of SIMD and buckets of b leads to non-trivial behaviors.

[2] P M Martini and W A Burkhard. Double hashing with multiple passbits. IJCS, 14(06):1165–1182, 2003.

Thank you!