Closing the Gap

Optimizing the Gap-Enumeration Algorithm

Michael Yantosca

University of Houston

April 28, 2018

Priority-Based Functional Reactive Programming (P-FRP)

- Pros
  - atomicity by function call
  - stronger invariants
  - easier application development
    - "Beautiful Concurrency" (https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/beautiful.pdf)
  - functional programming often feeds new features into other programming paradigms
- Cons
  - wasted computation time (abort)
  - difficult to analyze response time
  - harder system implementation
    - "TQueue can lead to thread starvation" (https://ghc.haskell.org/trac/ghc/ticket/9539 - opened 4 years ago and still unresolved!)
  - largely unproven in RTS

Worst-Case Response Time (WCRT) Analysis

- Time-Accurate Simulation (TAS)
  - straightforward
  - costly in time and space
- Gap-Enumeration Algorithm
  - based on red-black trees[2, pp. 263-280]
  - lower cost than TAS[1, pp. 14-16]

Gap-Enumeration Algorithm[1, p. 11]

1: **function** GAP-ENUMERATE-DYNAMIC(Γ_n, τ_j, w)
2: $L \leftarrow \lceil \frac{P_j}{w} \rceil$
3: $U \leftarrow P_j + \lceil \frac{P_j}{w} \rceil$
4: **while** $L < U$ **do**
5: $\sigma_n(P|_0^L) \leftarrow \{[0, L)\}$
6: **for** $i \leftarrow n-1, j$ **do**
7: $\sigma_{i-1}(P|_0^L) \leftarrow$ GAP-TRANSFORM($L, \sigma_i(P|_0^L), \Gamma_n, j$)
8: **if** $\sigma_{i-1}(P|_0^L) = \emptyset$ **then**
9: **return** -1
10: **end if**
11: **end for**
12: $[t_1, t_2) \leftarrow$ GAP-SEARCH($\sigma_j(P|_0^L), C_j$)
13: **if** $t_1 \geq 0$ **then**
14: $RT_j \leftarrow t_1 + C_j$
15: **end if**
16: **if** $RT_j < P_j$ **then**
17: **return** RT_j
18: **end if**
19: $L \leftarrow L + \lceil \frac{P_j}{w} \rceil$
20: **end while**
21: **return** -1
22: **end function**

Gap-Transformation Function[1, p. 12]

function GAP-TRANSFORM(W, $\sigma_i(P|_0^L)$, Γ_n, j)
$J_i \leftarrow \lceil \frac{W - R_j}{P_j} \rceil$
for $q \leftarrow 1, J_i$ **do**
$t \leftarrow R_j + P_j(q-1)$
$kgap \leftarrow$ MIN-GAP($\sigma_i(P|_0^L)$)
$t_1 \leftarrow entry[kgap]$
$t_2 \leftarrow exit[kgap]$
while $kgap \neq$ NIL($\sigma_i(P|_0^L)$) **do**
if $t_1 > t + P_j$ **then**
return $\emptyset$
end if
if $t < t_1$ **then**
$t \leftarrow t_1$
end if
if $t_1 \leq t$ and $t < t_2$ **then**
GAP-DELETE($\sigma_i(P|_0^L)$, $[t_1, t_2)$)
if $t + C_j = t_2$ **then**
GAP-INSERT($\sigma_i(P|_0^L)$, $[t_1, t)$)
exit while
end if
if $t + C_j < t_2$ **then**
GAP-INSERT($\sigma_i(P|_0^L)$, $[t_1, t)$)
GAP-INSERT($\sigma_i(P|_0^L)$, $[t + C_j, t_2)$)
exit while
end if
if $t + C_j > t_2$ **then**
GAP-INSERT($\sigma_i(P|_0^L)$, $[t_1, t)$)
end if
end if
if $t1 = entry[kgap]$ **then**
$kgap \leftarrow$ SUCCESSOR-GAP($kgap$)
end if
end while
end for
$\sigma_{i-1}(P|_0^L) \leftarrow \sigma_i(P|_0^L)$
return $\sigma_{i-1}(P|_0^L)$
end function

Gap-Search Algorithm[1, pp. 12-13]

- Effectively InOrder-Tree-Walk[2, pp. 245-246]
- O(n)
- Halts when gap is found that can fit the computational cost or gaps are exhausted

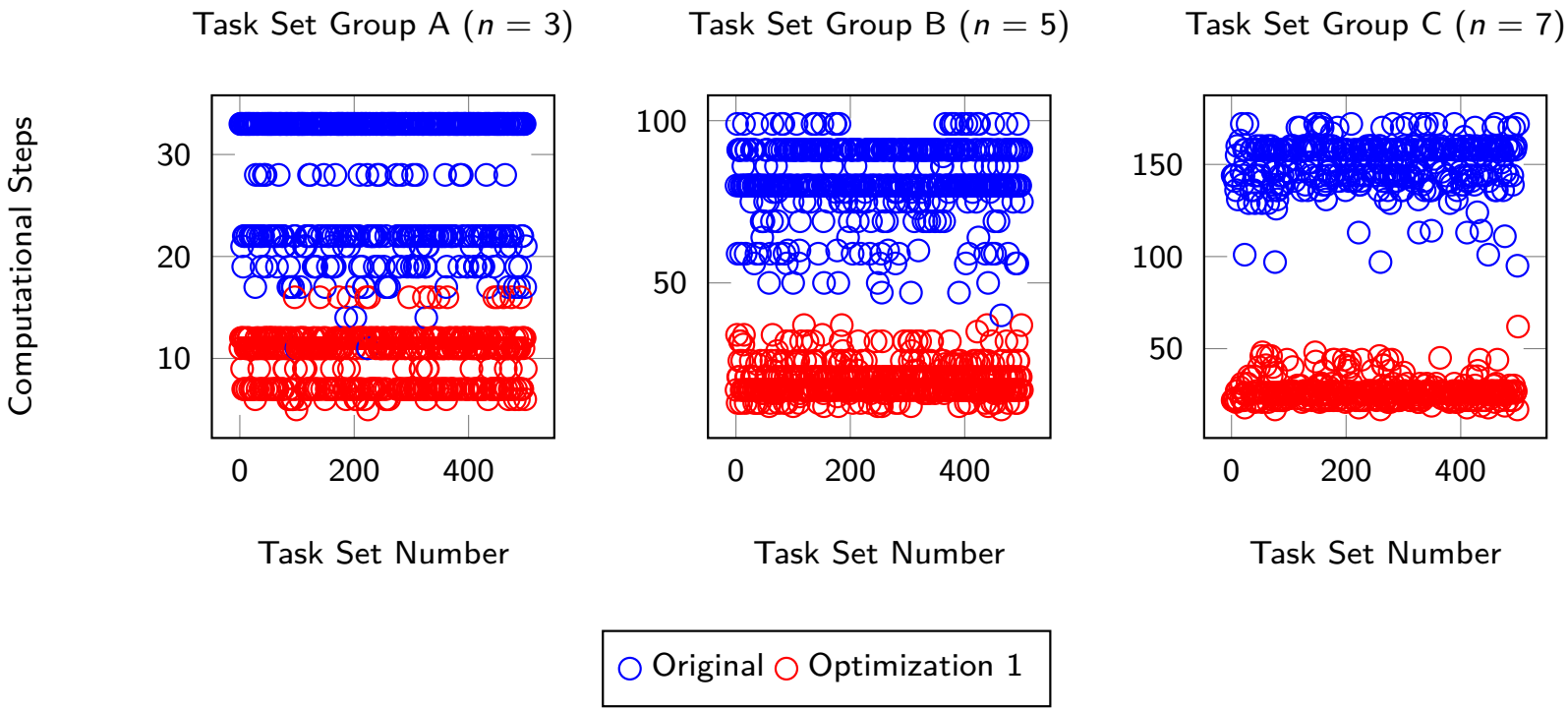
Room for Improvement

- Efficiency suggestions from the work[1, p. 17]
  - hash table to index location of k-gaps
  - 2-D array to keep track of gaps created per task
- Algorithmic improvements
  - stop reinserting gaps of the form $[t, t)$
  - gap re-write
  - gap splice

Optimization 1

function GAP-TRANSFORM-POSITIVE(W, $\sigma_i(P|_0^L)$, Γ_n, j)
  $J_i \leftarrow \lceil \frac{W-R_j}{P_j} \rceil$
  for $q \leftarrow 1, J_i$ **do**
    $t \leftarrow R_j + P_j(q-1)$
    $kgap \leftarrow$ MIN-GAP($\sigma_i(P|_0^L)$)
    $t_1 \leftarrow entry[kgap]$
    $t_2 \leftarrow exit[kgap]$
    while $kgap \neq$ NIL($\sigma_i(P|_0^L)$) **do**
      if $t_1 > t + P_j$ **then**
        return $\emptyset$
      end if
      if $t < t_1$ **then**
        $t \leftarrow t_1$
      end if
      if $t_1 \leq t$ and $t < t_2$ **then**
        GAP-DELETE($\sigma_i(P|_0^L)$, $[t_1, t_2)$)
        if $t + C_j = t_2$ **then**
          if $t > t_1$ **then**
            GAP-INSERT($\sigma_i(P|_0^L)$, $[t_1, t)$)
          end if
          exit while
        end if
        if $t + C_j < t_2$ **then**
          if $t > t_1$ **then**
            GAP-INSERT($\sigma_i(P|_0^L)$, $[t_1, t)$)
          end if
          GAP-INSERT($\sigma_i(P|_0^L)$, $[t + C_j, t_2)$)
          exit while
        end if
        if $t + C_j > t_2$ **then**
          if $t > t_1$ **then**
            GAP-INSERT($\sigma_i(P|_0^L)$, $[t_1, t)$)
          end if
        end if
      end if
      if $t1 = entry[kgap]$ **then**
        $kgap \leftarrow$ SUCCESSOR-GAP($kgap$)
      end if
    end while
  end for
  $\sigma_{i-1}(P|_0^L) \leftarrow \sigma_i(P|_0^L)$
  return $\sigma_{i-1}(P|_0^L)$
end function

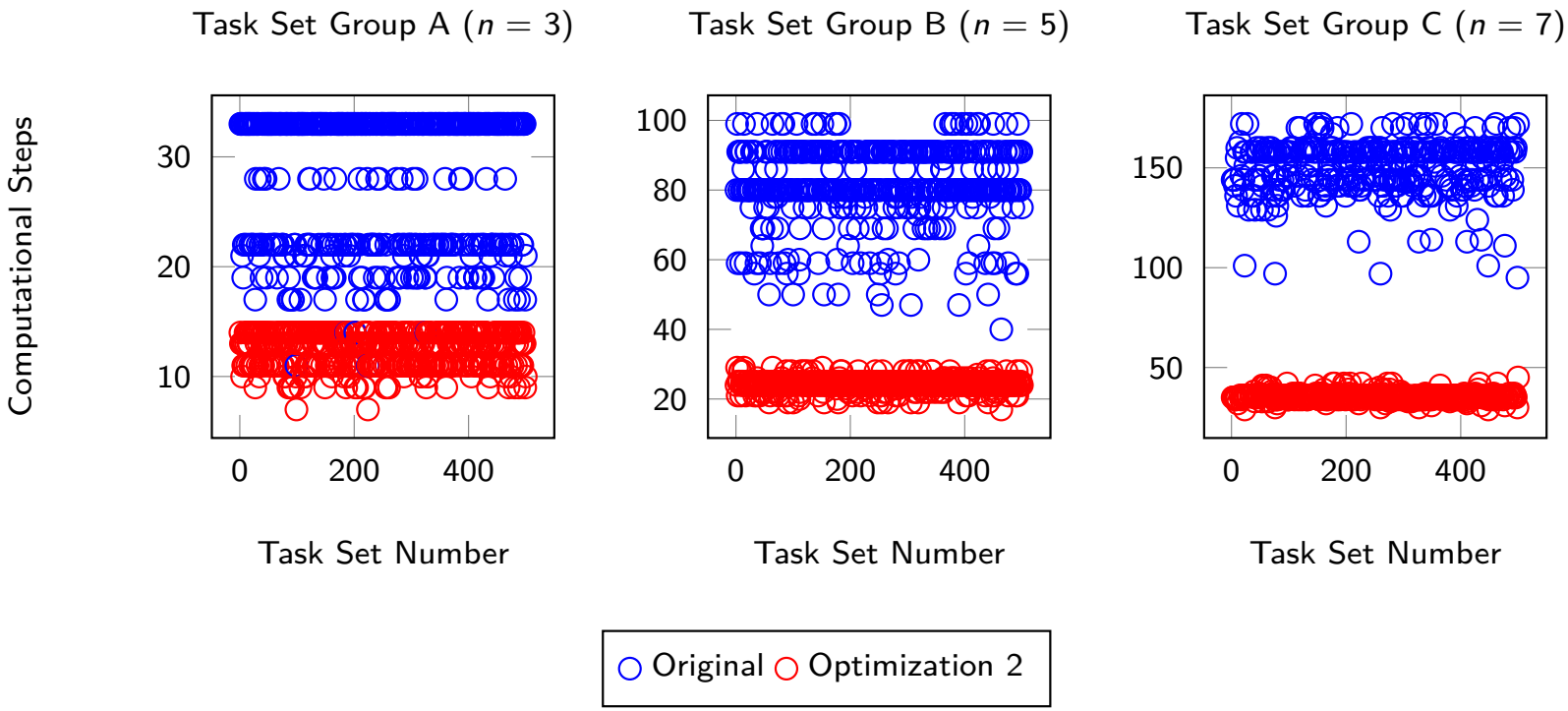
Original vs. Optimization 1



Optimization 2

```
function GAP-TRANSFORM-POSITIVE-LIST(W, σ_i(P|_0^L), Γ_n, j)
    J_i ← ⌈(W − R_j)/P_j⌉
    for q ← 1, J_i do
        t ← R_j + P_j(q − 1)
        kgap ← HEAD(σ_i(P|_0^L))
        t_1 ← entry[kgap]
        t_2 ← exit[kgap]
        while kgap ≠ TAIL(σ_i(P|_0^L)) do
            if t_1 > t + P_j then
                return ∅
            end if
            if t < t_1 then
                t ← t_1
            end if
            if t_1 ≤ t and t < t_2 then
                if t + C_j ≥ t_2 then
                    if t > t_1 then
                        entry[kgap] ← t_1
                        exit[kgap] ← t
                    else
                        SPLICE-OUT(σ_i(P|_0^L), kgap)
                    end if
                end if
                if t + C_j = t_2 then
                    exit while
                end if
            end if
            if t + C_j < t_2 then
                entry[kgap] ← t + C_j
                exit[kgap] ← t_2
                if t > t_1 then
                    SPLICE-IN(σ_i(P|_0^L), kgap, [t_1, t))
                end if
                exit while
            end if
        end if
        if t1 = entry[kgap] then
            kgap ← NEXT(kgap)
        end if
    end while
end for
σ_{i−1}(P|_0^L) ← σ_i(P|_0^L)
return σ_{i−1}(P|_0^L)
end function
```

Original vs. Optimization 2



Conclusions and Future Work

- Space cost is time cost.
- Stream processing techniques can reduce costs (minimize passes over data).
- Pointers are real (helpful).
- The gap search function needs work.
  - It will probably require some sort of time tradeoff with space.
- More reuse of computational results would be nice.
  - e.g., reusing gap lists from previous iterations in next iteration

References

Chaitanya Belwal and Albert M. K. Cheng. *Determining Actual Response Time in P-FRP*. Technical Report UH-CS-10-05. University of Houston Computer Science Department, June 28, 2010.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press and McGraw-Hill Book Company, 1997. ISBN: 0-262-03141-8.