Searching for the Optimal Data Partitioning Shape for Parallel Matrix Matrix Multiplication on 3 Heterogeneous Processors

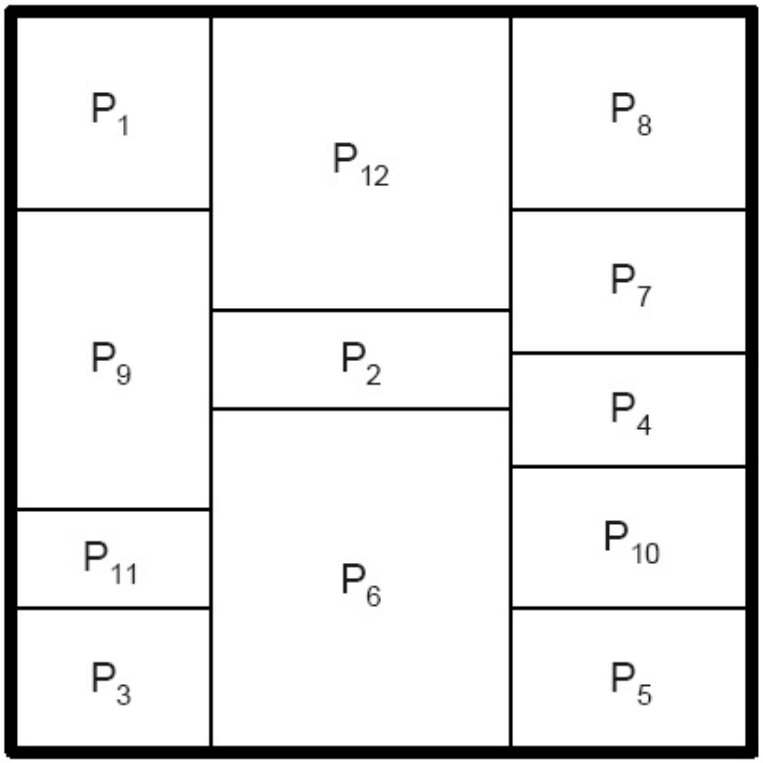
Ashley DeFlumere, Alexey Lastovetsky

Heterogeneous Computing Laboratory, University College Dublin, Dublin, Ireland

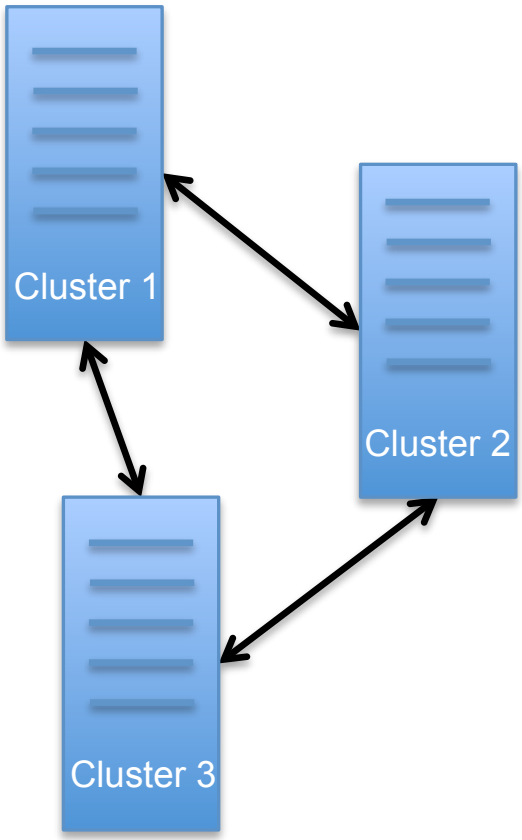
Heterogeneity in Computing Workshop, IPDPS, May 2014

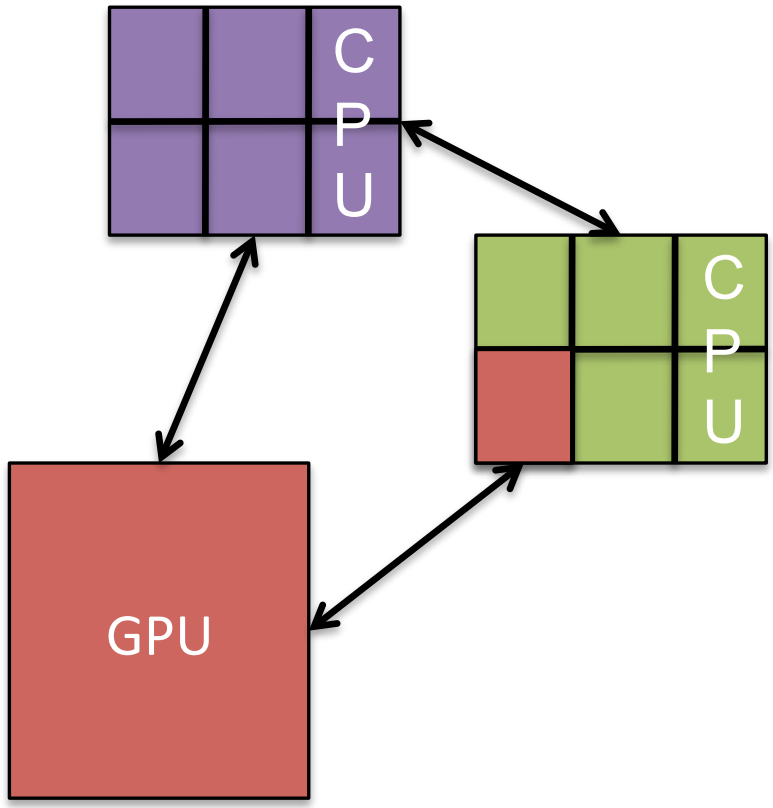
What is Optimal Data Partitioning?

- Divide the matrices' data among available processors to minimize execution time
- Finding the optimal data partition for an arbitrary number of heterogeneous processors is an NP-complete problem



- Focus on 3 abstract processors as a model for interacting clusters or GPU systems





Modelling MMM - Assumptions

Define the problem,

Computation

- Each Matrix A, B, C is square and identically partitioned
- Each Processor has a defined computation speed, expressed as a ratio $P_r : R_r : S_r$, where $S_r = 1$
- Five different MMM Algorithms used to create models of execution time

Modelling MMM - Assumptions

Define the problem,

Computation

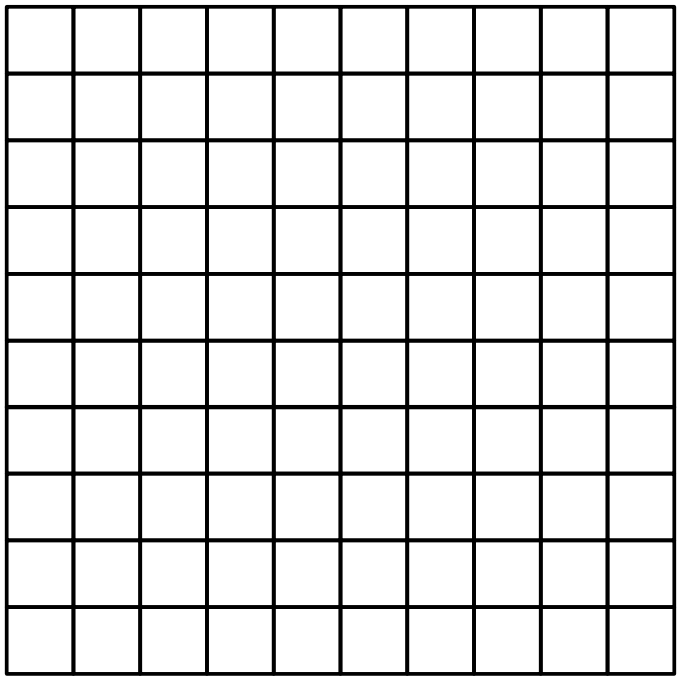
- Each Matrix A, B, C is square and identically partitioned
- Each Processor has a defined computation speed, expressed as a ratio $P_r : R_r : S_r$, where $S_r = 1$
- Five different MMM Algorithms used to create models of execution time

Communication

- Modelled by Hockney, $\alpha + \beta \times M$
- Each Processor communicates with both other processors, and all three links are of the same speed (other topologies are described briefly later)

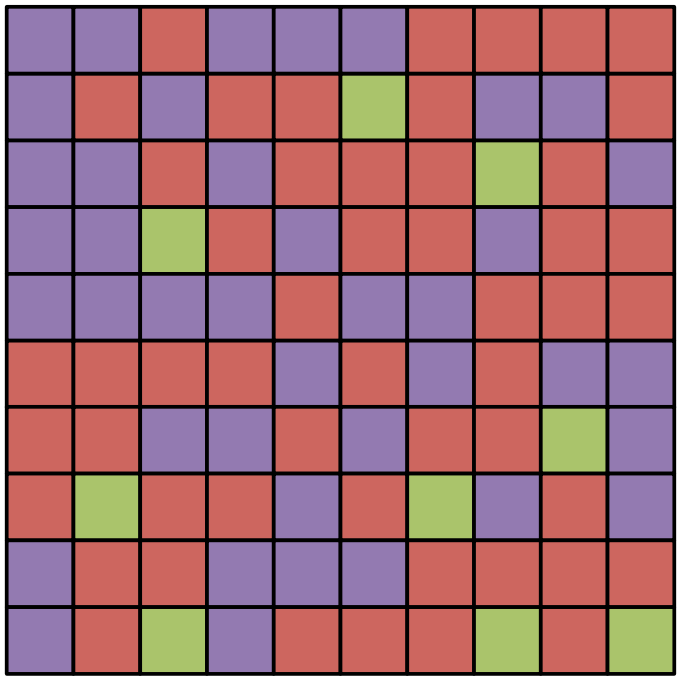
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



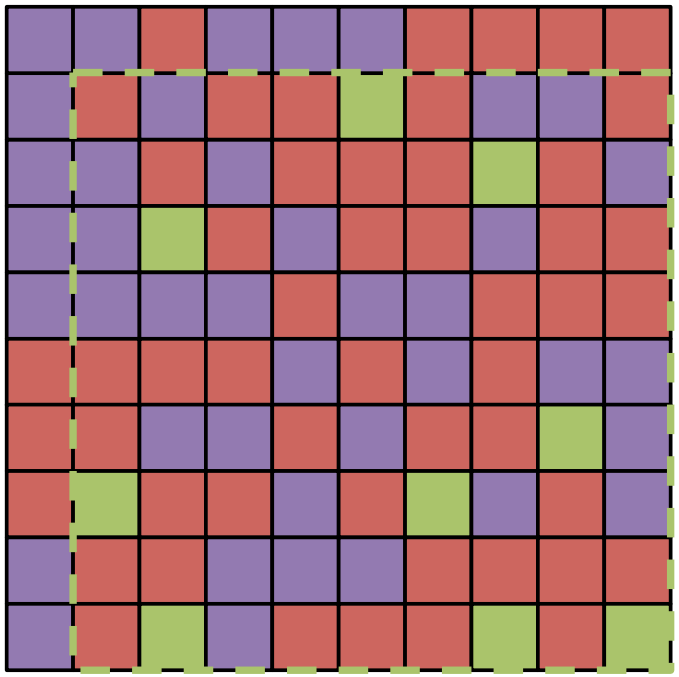
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



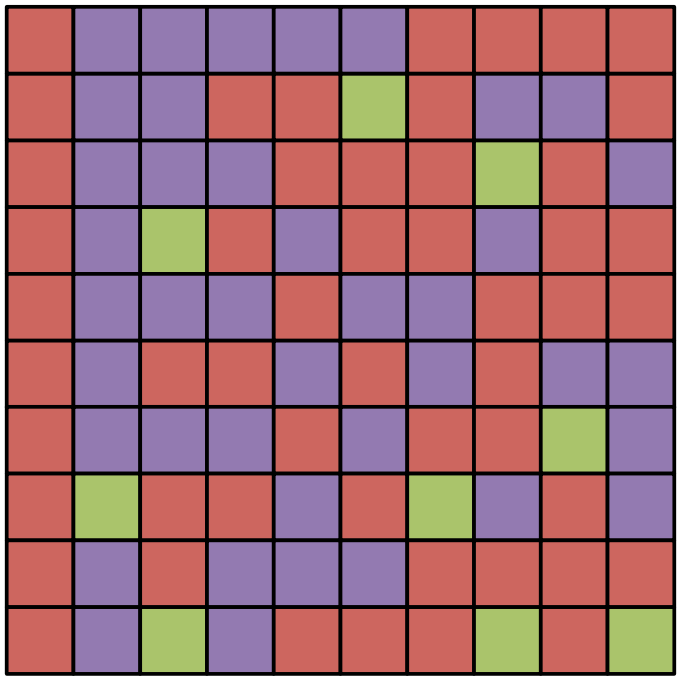
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



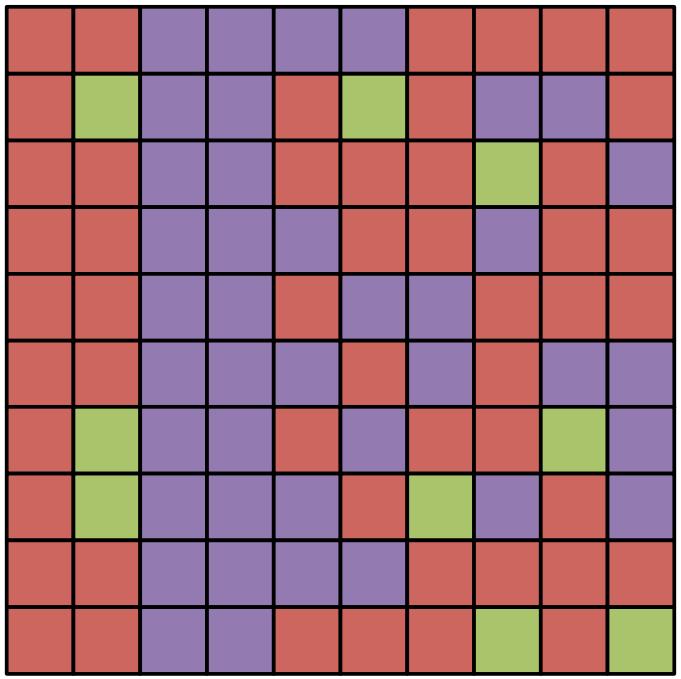
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



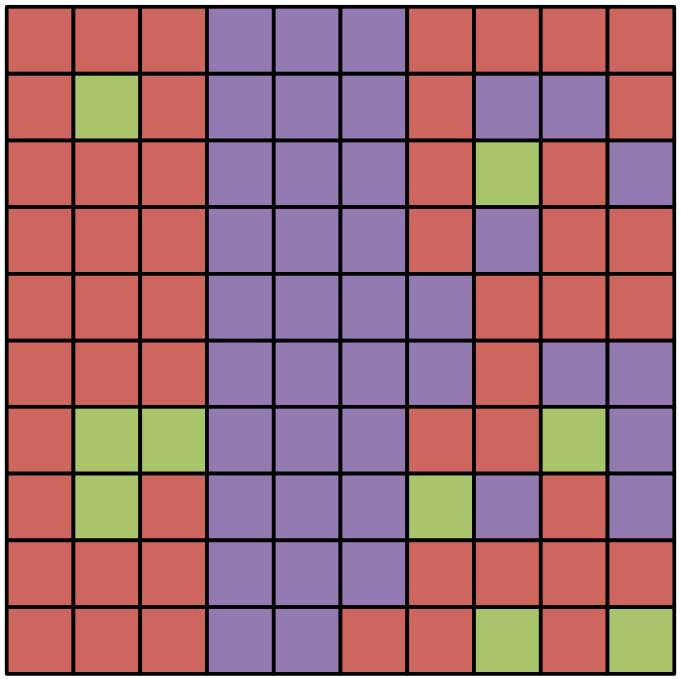
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



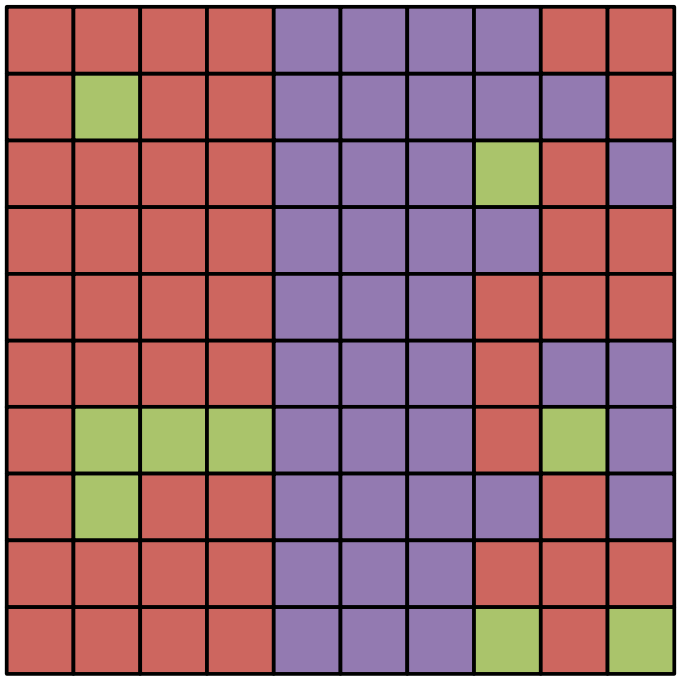
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



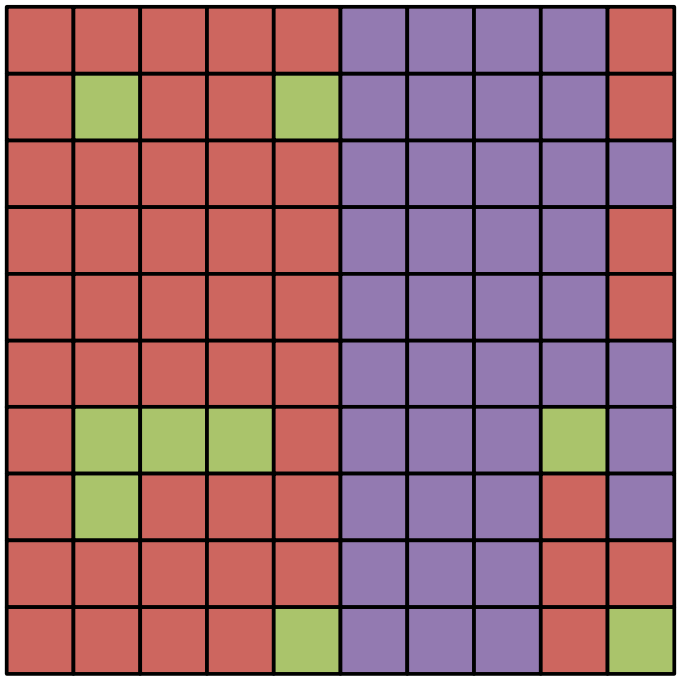
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



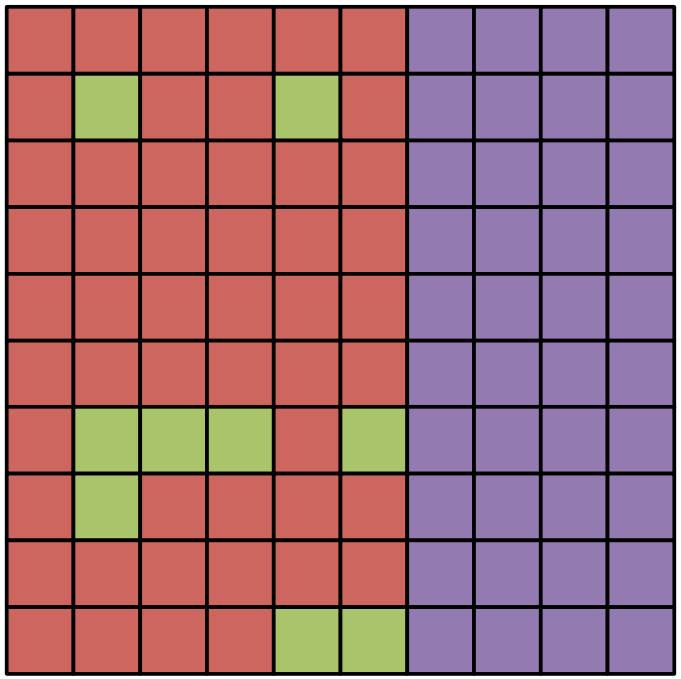
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



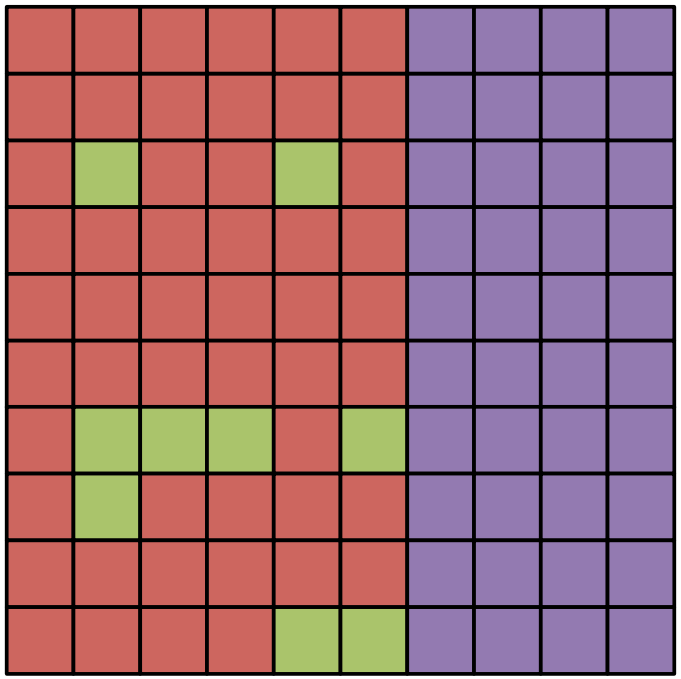
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



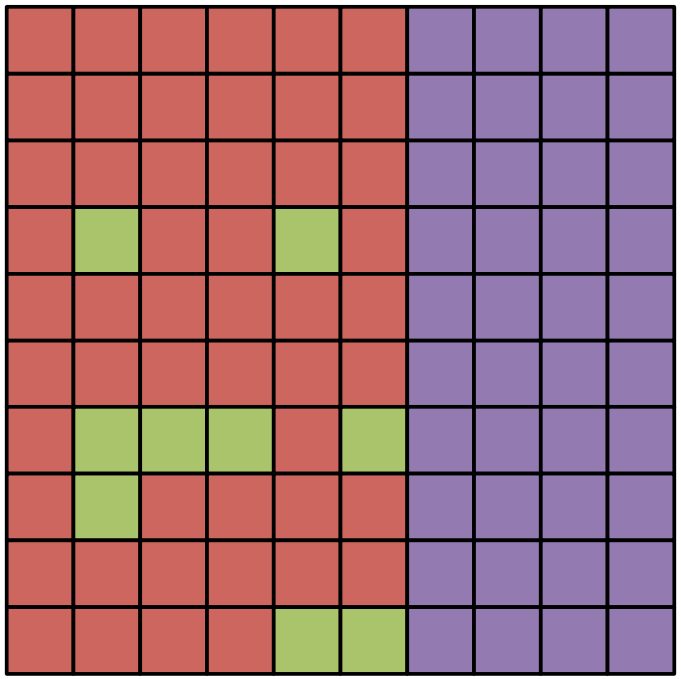
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



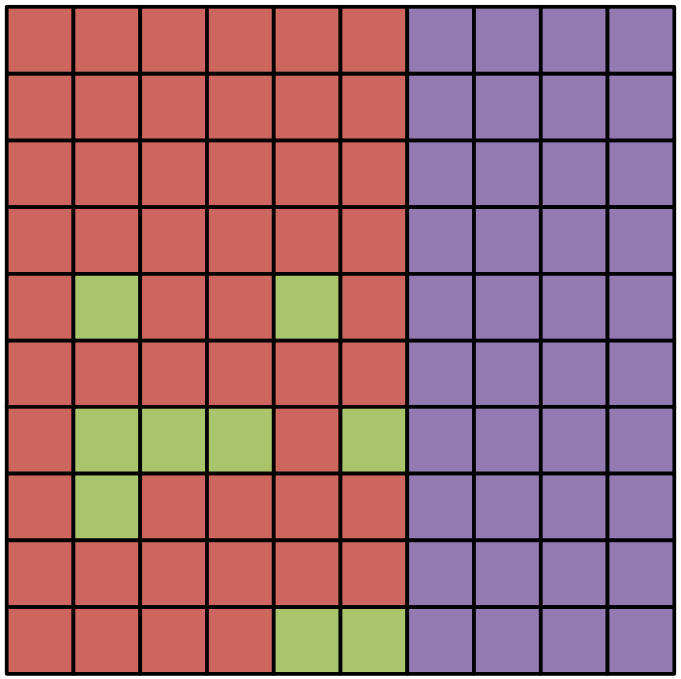
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



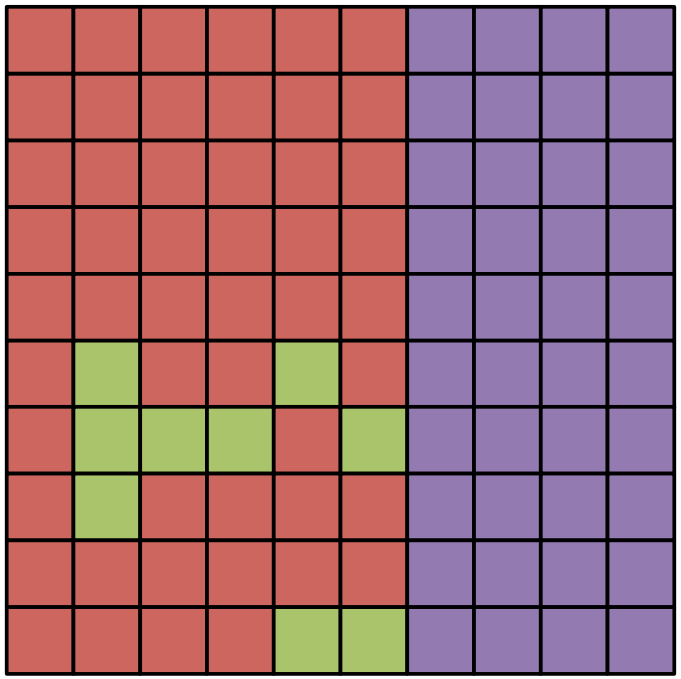
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



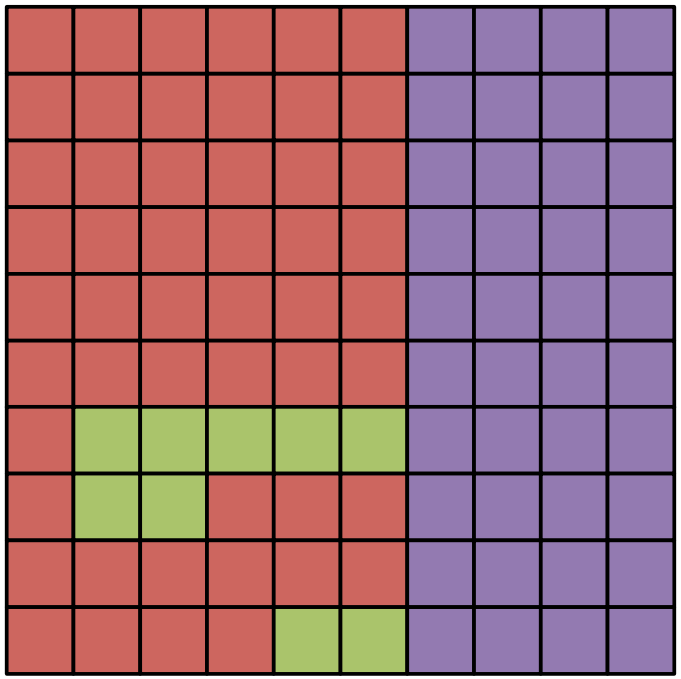
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



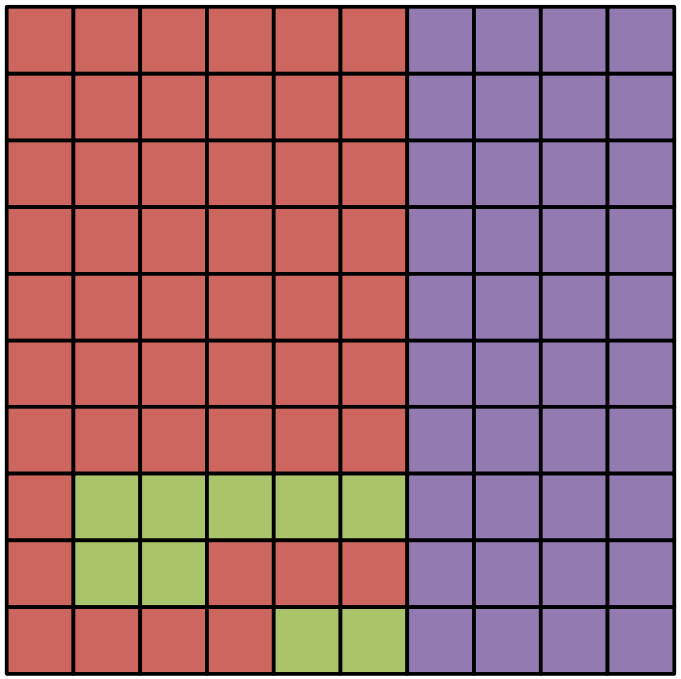
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



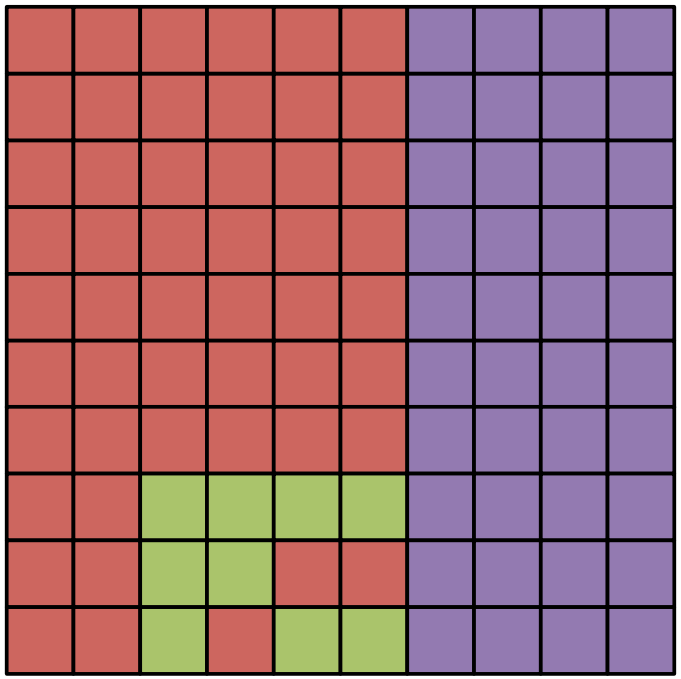
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



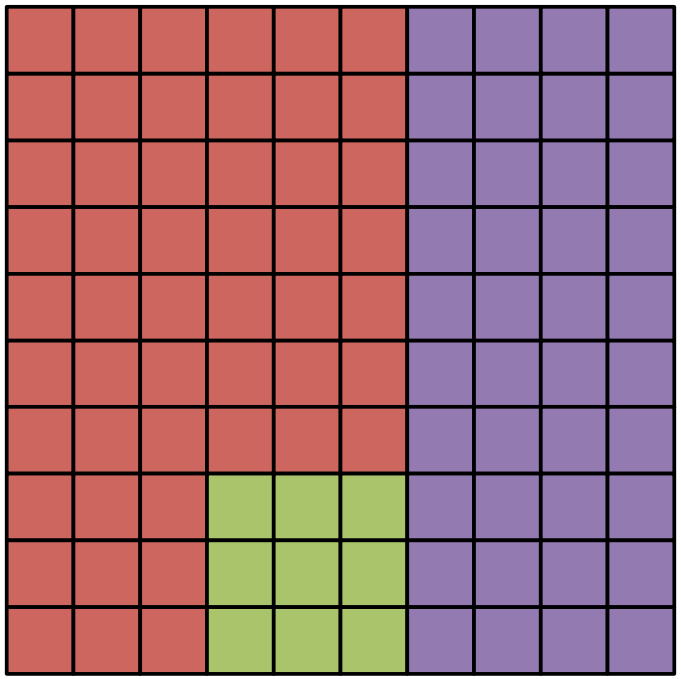
Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



Finding Partition Shapes

- *Partition Shape* is a visual form of the formal partition equation, $q(i, j)$
- *Push Operation* transforms a partition shape, decreasing (or leaving unchanged) the volume of communication and execution time of the shape



DFA Program

Three Processor Challenges

- Two Processor Push can be mathematically shown to always converge to recognizable shapes
- Three Processor Push is more complex
- Consider legality of moving both processors, not simply the active processor being Pushed
- Must show that Three Processor Push always forms some recognizable shape

DFA Program Definition

- Present problem as a Deterministic Finite Automaton, $(Q, \Sigma, \delta, q_0, F)$

DFA Program Definition

- Present problem as a Deterministic Finite Automaton, $(Q, \Sigma, \delta, q_0, F)$
- Q - the finite set of states, possible data partition shapes
- Σ - the finite set of the alphabet, the processors and directions of Push
- δ - $Q \times \Sigma \rightarrow Q$, the transition function, the Push operation
- q_0 - the start state, chosen at random
- F - $F \subseteq Q$, the accept states, candidates to be the optimum

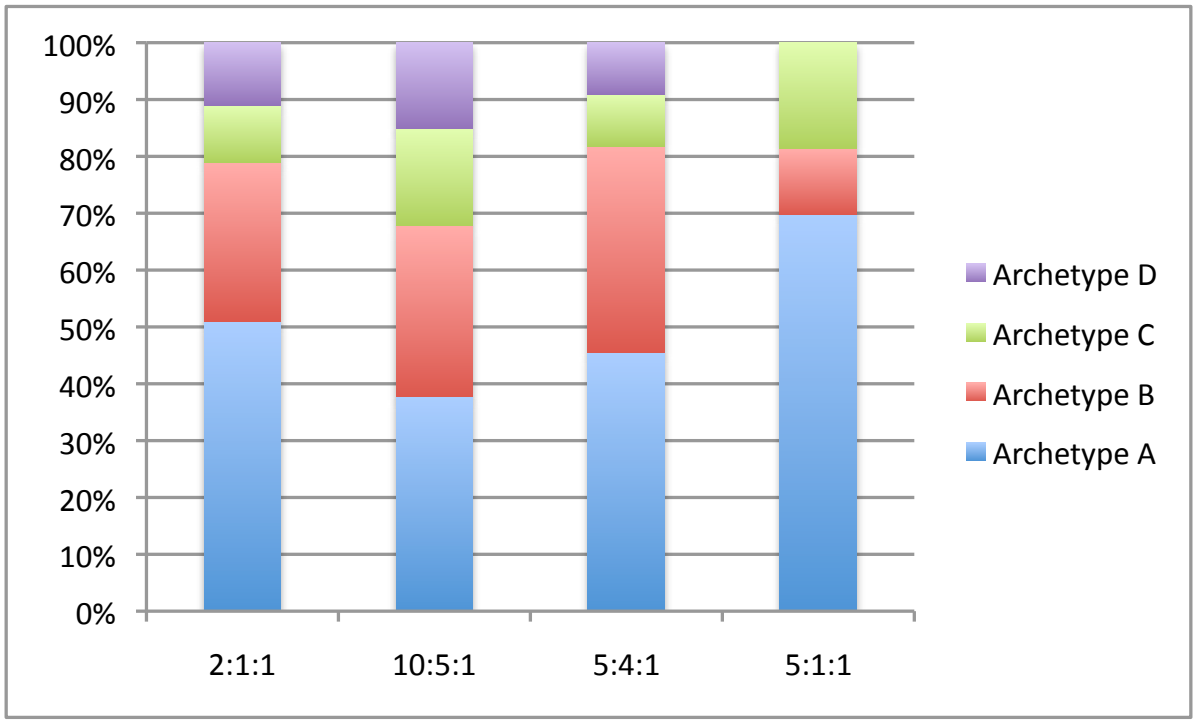
Push Transition Algorithm, δ

Initialize $q_1 \leftarrow q$
$(g, h) \leftarrow (r_{top} + 1, r_{left})$
for $j = r_{left} \rightarrow r_{right}$ **do**
  if $q(r_{top}, j) = 0$ **then**
    {Element is dirty, clean it}
    $(g, h) \leftarrow$ find (g, h) {Function defined below}
    if $q(g, h) = 1$ **then**
      $q_1(r_{top}, j) \leftarrow 1$ {Cleaned element assigned to S}
    end if
    if $q(g, h) = 2$ **then**
      $q_1(r_{top}, j) \leftarrow 2$ {Cleaned element assigned to P}
    end if
    $q_1(g, h) \leftarrow 0$ {Put displaced element in new spot}
  end if
  $j \leftarrow j + 1$
end for

findTypeOne(g, h) {Look for a suitable slot to put element}
for $g \rightarrow r_{bottom}$ **do**
  for $h \rightarrow r_{right}$ **do**
    if $q_1(g, h) \neq 0$ &&
    $(row(q, r_{top}, (q_1(g, h)) = 1$
    $\parallel col(q, j, (q_1(g, h)) = 1)$ &&
    $(row(q, g, R) = 1 \parallel col(q, h, R) = 1$
    then
      return (g, h)
    end if
    $h \leftarrow h + 1$
  end for
  $h \leftarrow k_{left}$
  $g \leftarrow g + 1$
end for
return $q_1 = q$ {No Type One Push$\downarrow q(R)$ possible}

Experimental Setup

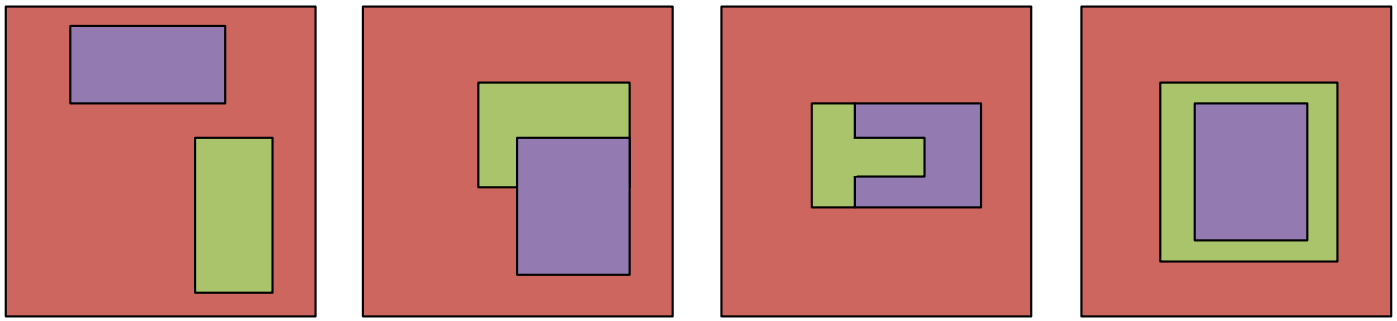
- Set $N = 1000$, use variety of ratios of $P_r : R_r : S_r$
- Run DFA program minimum 10,000 times per processor ratio



Analysis

Shape Archetypes

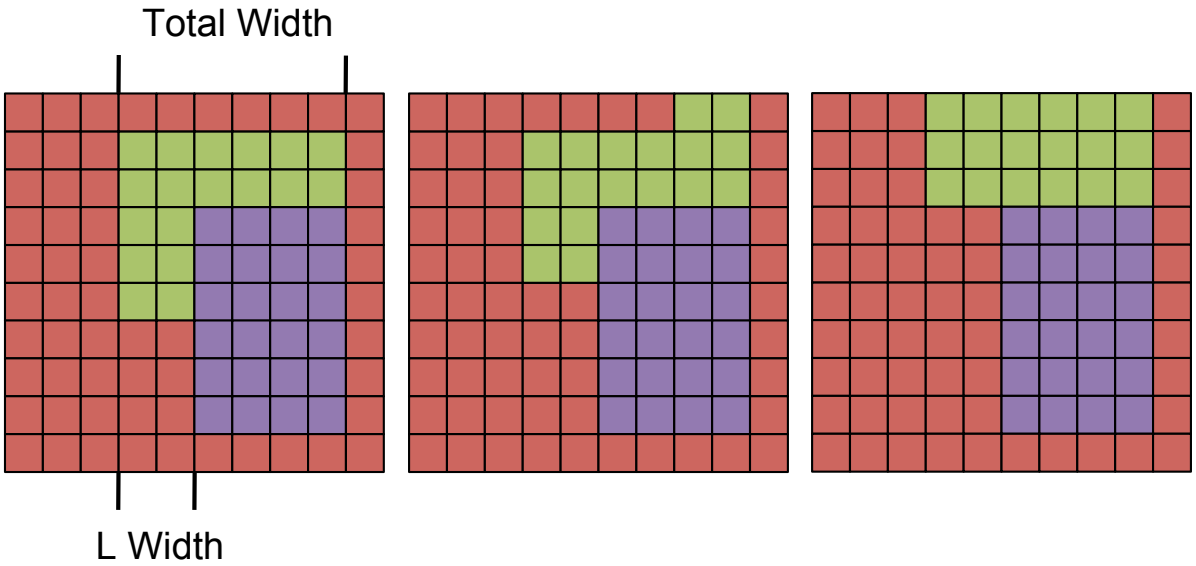
- Each output partition categorized by Enclosing Rectangles and number of Corners
- All output partitions fall into one of 4 Shape Archetypes



Analysis

Distilling to Shape Archetype A

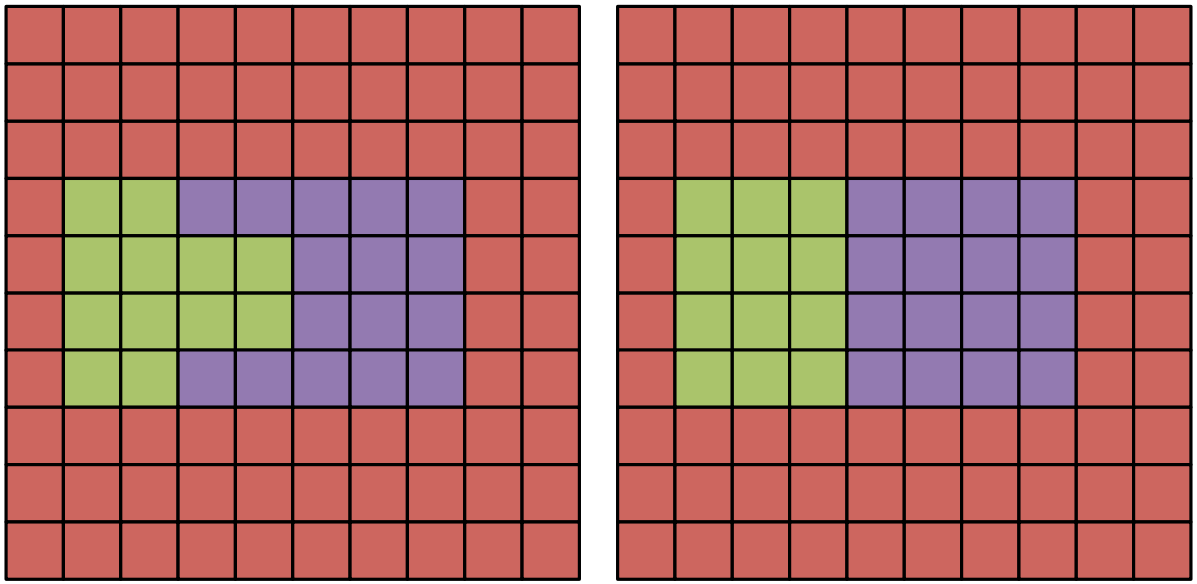
- All Shape Archetypes can be transformed to look like Archetype A, without increasing volume of communication
- Archetype B $\rightarrow$ Archetype A : move elements of "L" shape processor until a rectangle is formed



Analysis

Distilling to Shape Archetype A

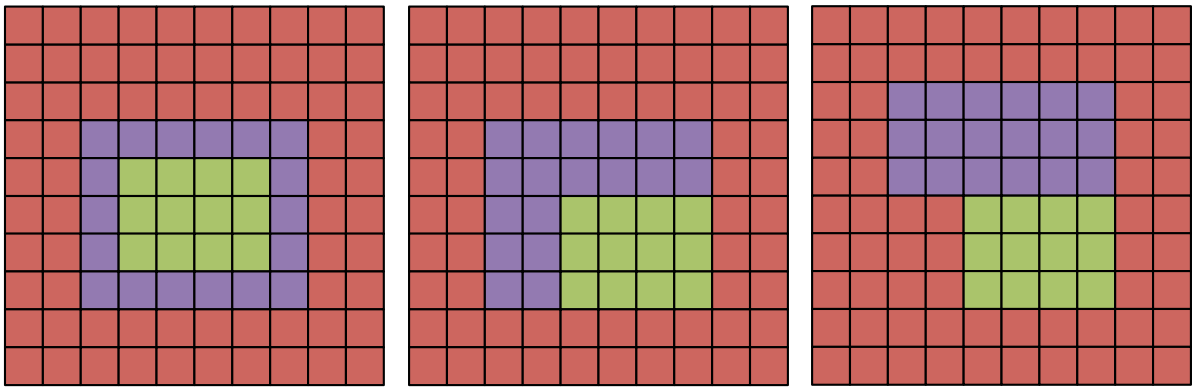
- All Shape Archetypes can be transformed to look like Archetype A, without increasing volume of communication
- Archetype C $\rightarrow$ Archetype A : use the Push operation



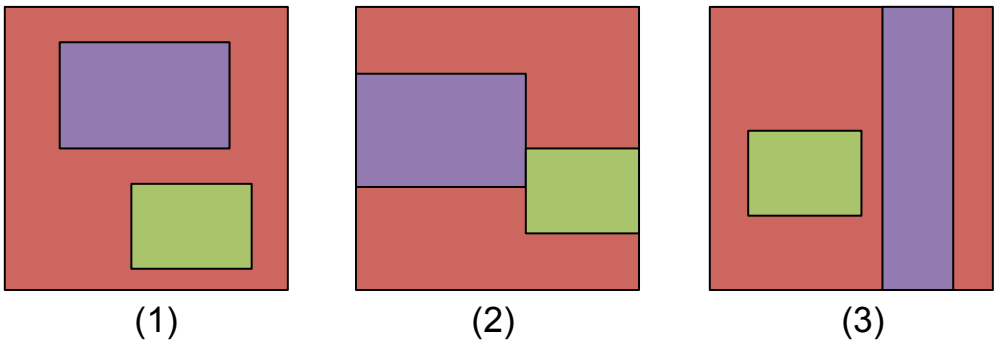
Analysis

Distilling to Shape Archetype A

- All Shape Archetypes can be transformed to look like Archetype A, without increasing volume of communication
- Archetype D $\rightarrow$ Archetype A : transform to Archetype B, then move elements of "L" shape processor



Analysis

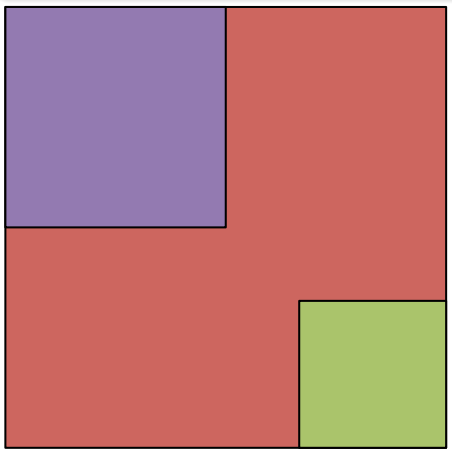


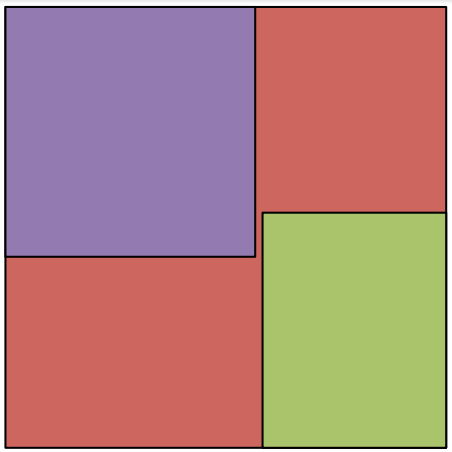
Examples of Archetype A partition shapes

Finding Canonical Shape

Type One Partition Shapes

- Two processors assigned non-overlapping rectangles, of combined length and height less than N
- The optimal size to of a rectangle is a square (minimizes sum of half-perimeters, and so volume of communication)
- What if Processors R and S are assigned too many elements to be non-overlapping squares?





$$P_r + R_r + S_r = T$$

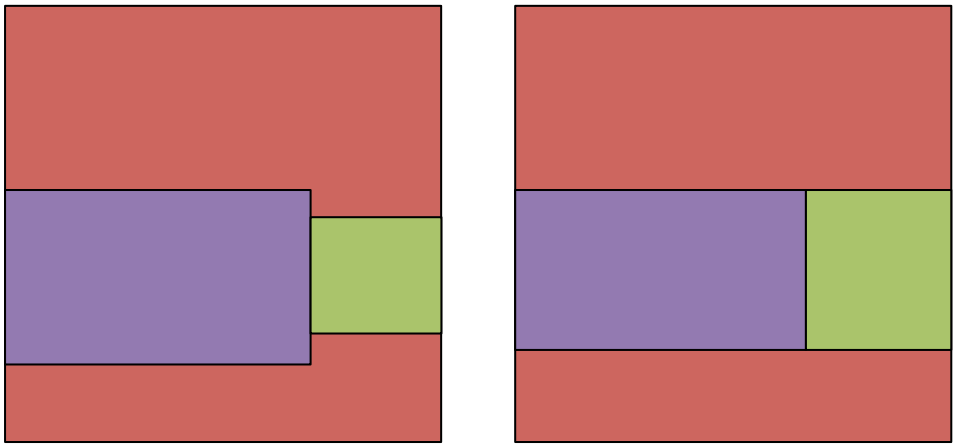
$$\sqrt{\frac{R_r}{T}} + \sqrt{\frac{S_r}{T}} < 1$$

$$2\sqrt{R_r} < P_r$$

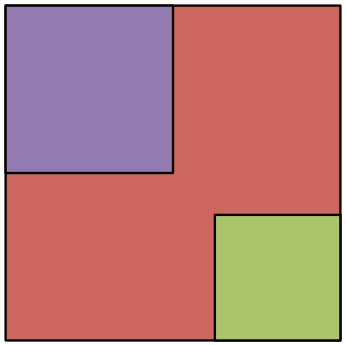
Finding Canonical Shape

Type Two and Four Partition Shapes

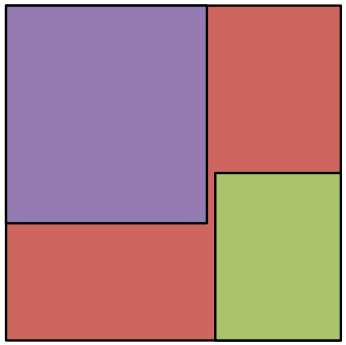
- Two processors assigned non-overlapping rectangles of combined length N
- Type Two - height of two processors not equal
- Type Four - height of two processors equal, always has a lower volume of communication than Type Two



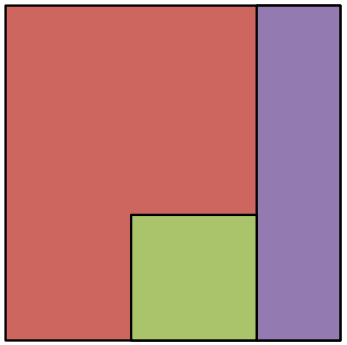
Finding Canonical Shape



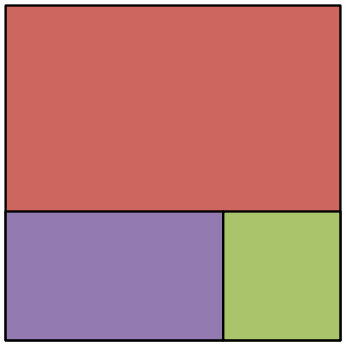
(1) Square Corner



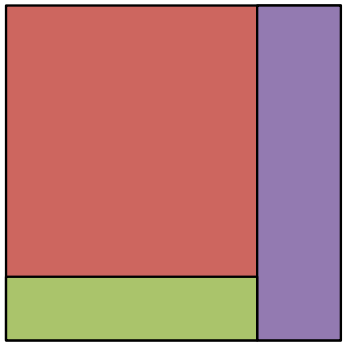
(2) Rectangle Corner



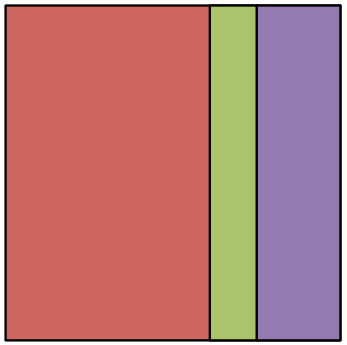
(3) Square Rectangle



(4) Block Rectangle



(5) L Rectangle



(6) Traditional Rectangle

Canonical versions of all candidate partition shapes

Conclusion

- Push DFA shows arbitrary arrangement of elements will condense to a recognizable three processor shape
- The shapes produced by Push DFA are reducible to a small set of candidate partitions
- The optimal data partitioning shape for all computational power and bandwidth ratios must be one of these six shapes
- Three of the six candidates are non-rectangular partition shapes

Thank You