О парадигме универсального языка параллельного программирования на примере задач SSSP и BFS

А.В. Климов
arkady.klimov@gmail.com

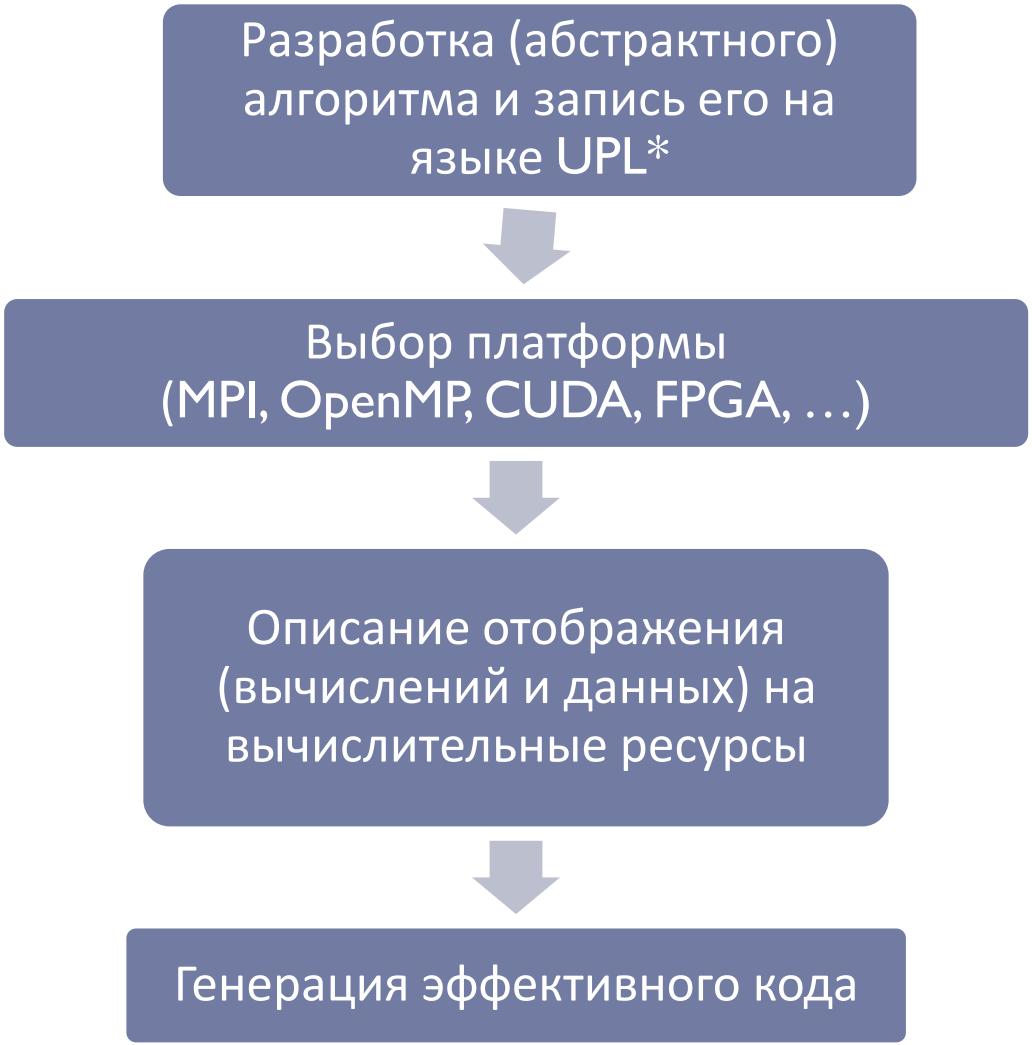
Институт проблем проектирования в микроэлектронике (ИППМ) РАН

GraphHPS-2017
2 марта
2017

Проблема параллельного программирвоания

- Параллельное программирование – трудное дело
- Существует множество разных параллельных архитектур и платформ
- Возможно ли иметь один язык для записи абстрактных алгоритмов и компилировать с него для всех платформ?
- Это парадигма WORE (или WOCA): Write-Once-Run-Everywhere (-Compute-Anywhere)
- Должно допускаться непосредственное исполнение (быть может неэффективное)
- Возможны дополнительные средства отображения
- Какие есть в мире предложения на эту тему?

Стадии разработки – что хотелось бы в идеале



*UPL – Universal Parallel Language

Существующие WORE проекты

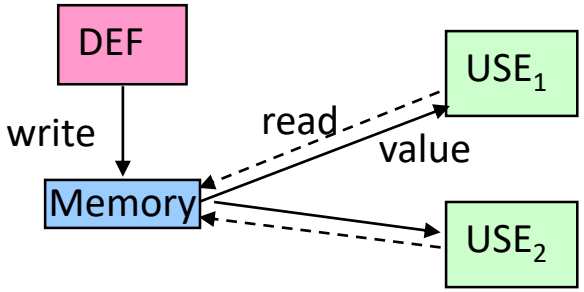
(все они так или иначе основаны на потоковой модели вычислений)

Имя (кто, где, когда)	Особенности	Применение
LabVIEW (National Instruments, USA, 1986)	Графический язык программирования. Есть компиляция для FPGA.	Ограниченная область применения (измерения и управление)
Single Assignment C (SAC) (University of Kiel, Germany, 1994)	Функциональный язык, синтаксис как C, строгая типизация, вывод типов, shape-инвариантность, выражения в стиле map и fold.	Работа с плотными массивами
Concurrent Collections (CnC) и его производные: DFGR, PIPES. Изначально: TStreams	Двудольный статический граф с явными тегами (индексами). Данные и операции отдельно. Отдельные средства отображения.	Вычисления с детерминированным деревом задач. Парадигма «сбора».
TensorFlow (Google)	Библиотека операций над плотными многомерными массивами – тензорами	Deep Neural Networks
Полиэдральная модель для линейных вложенных циклов	Входной язык – обычный C или фортран (подмножество). Средств отображения нет (можно по виткам циклов).	Автоматическое распараллеливание на OpenMP, CUDA (для квази-аффинных программ). Распределение и отображение выбирается автоматически.
UPL(G) (ИППМ РАН, 2016)	Динамический потоковый язык в парадигме «раздачи» с явными индексами. Графический язык. Отдельные средства отображения	Распределенная обработка с нерегулярными и разреженными данными
Charm++ plus Charisma	Перемещаемые объекты (агенты) с состояниями, односторонние вызовы, Charisma – orchestrating language	То же. RTS своя (не WORE).

Парадигмы сбора и раздачи

Все традиционные языки основаны на парадигме сбора

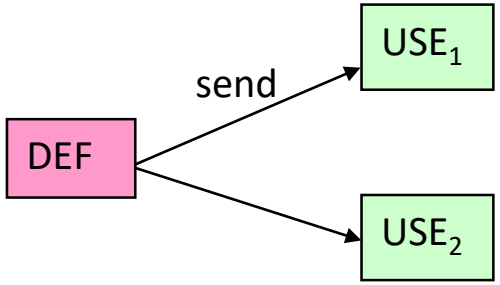
Парадигма СБОРА
(C, Fortran, etc)



Поставщик сохраняет свой результат в памяти по некоторому адресу, откуда его запрашивают потребители по мере надобности (по тому же адресу)

Обычно результат сохраняется рядом с производителем – принцип owner computes

Парадигма РАЗДАЧИ
(dataflow, message passing, CHARM++)



Поставщик знает (вычисляет) сам «адреса» всех потребителей, на которые и рассылает свой результат.

Для распределенного вычисления парадигма раздачи лучше:
одна передача вместо двух, нет ожидания при чтении

Схема алгоритма SSSP. Наивный вариант (без синхронизации)

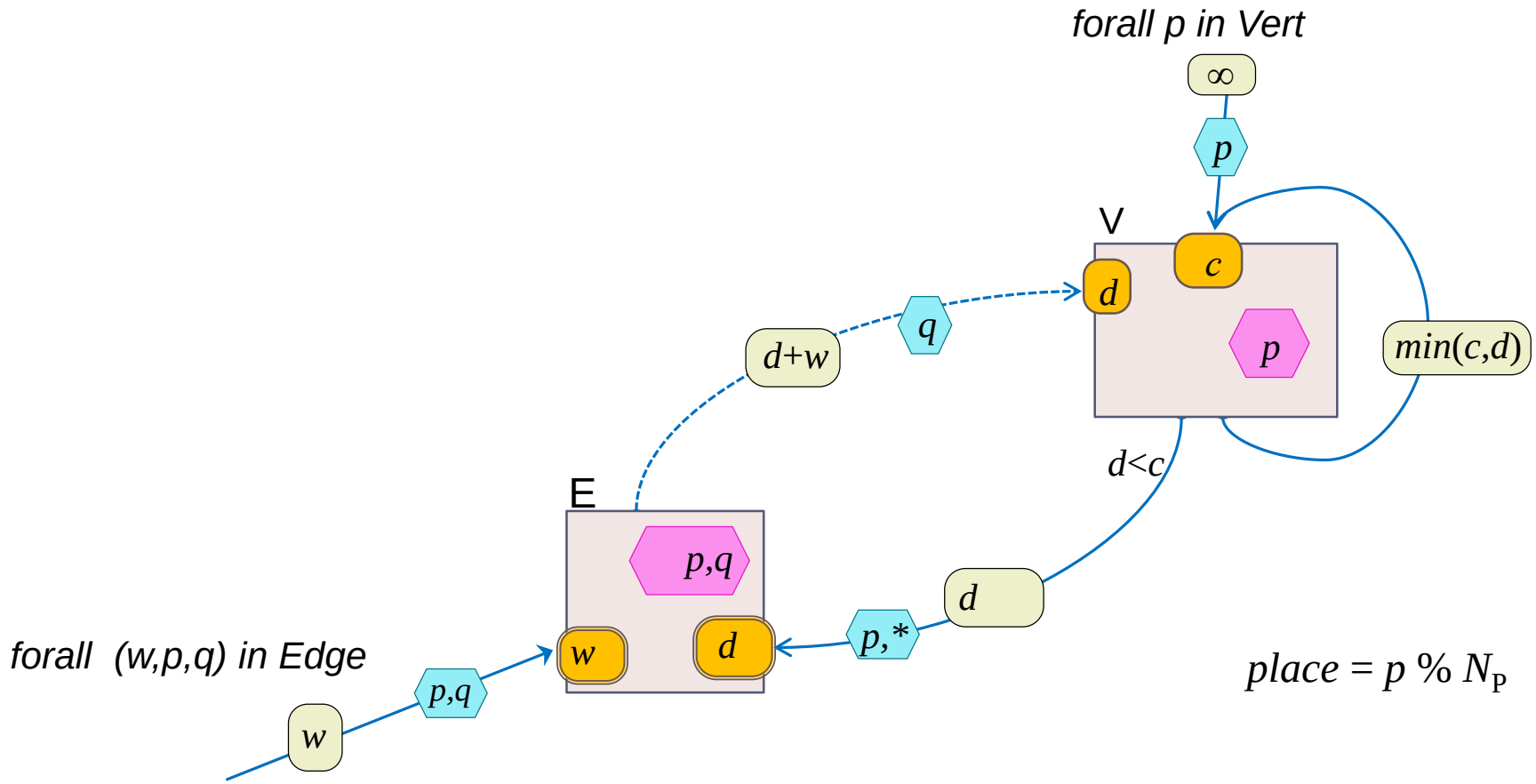
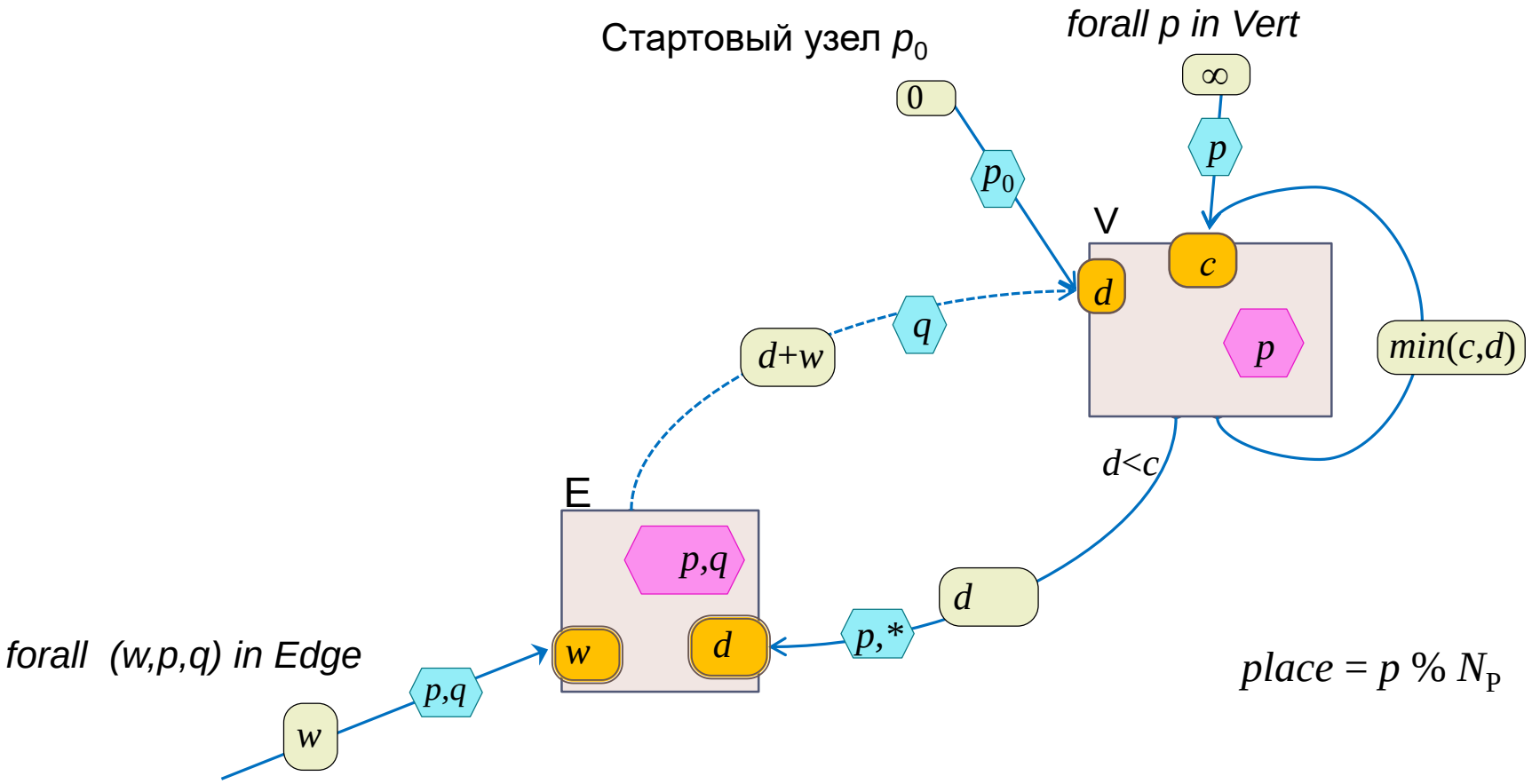
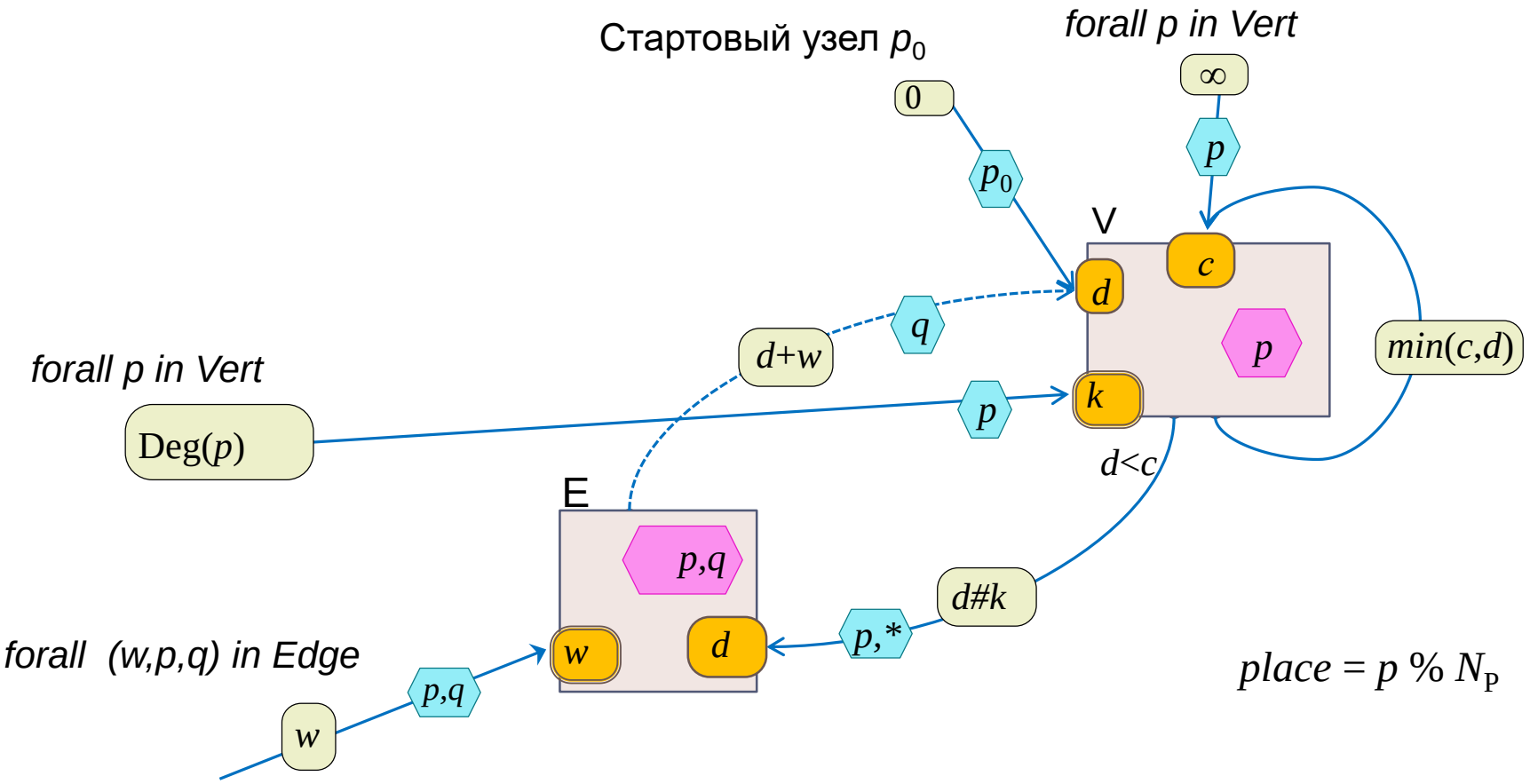


Схема алгоритма SSSP. Наивный вариант (без синхронизации)



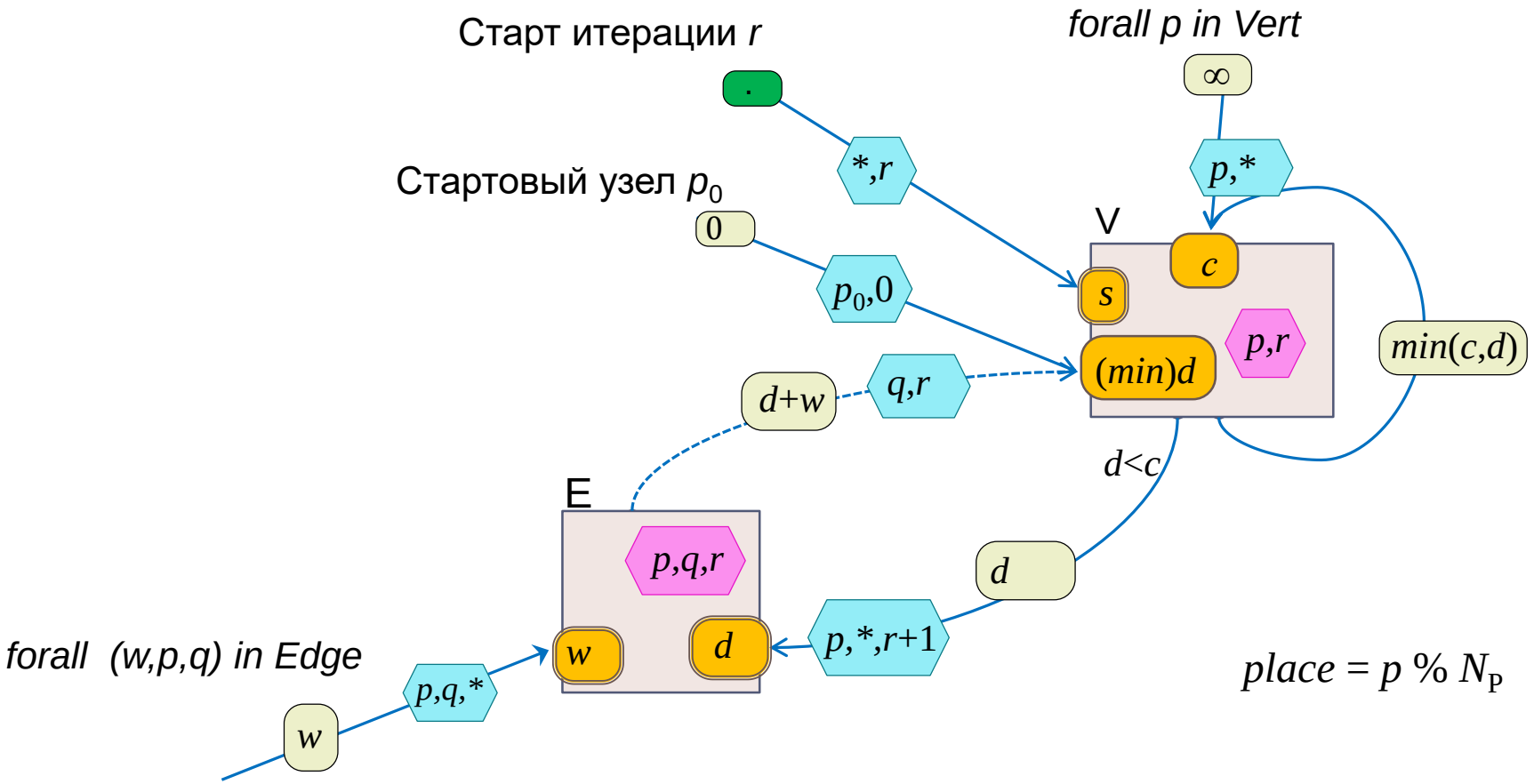
По завершению всякой активности кратчайшие расстояния от p_0 до p, будут находиться в V.c<p> . Возможны многократные проходы по дугам.
Пунктиром обозначены передачи по дугам. Только они могут быть нелокальными.
Узлы распределяются по ядрам по полю p взятием остатка от деления на число ядер N_P.

Схема алгоритма SSSP. Наивный вариант (без синхронизации)



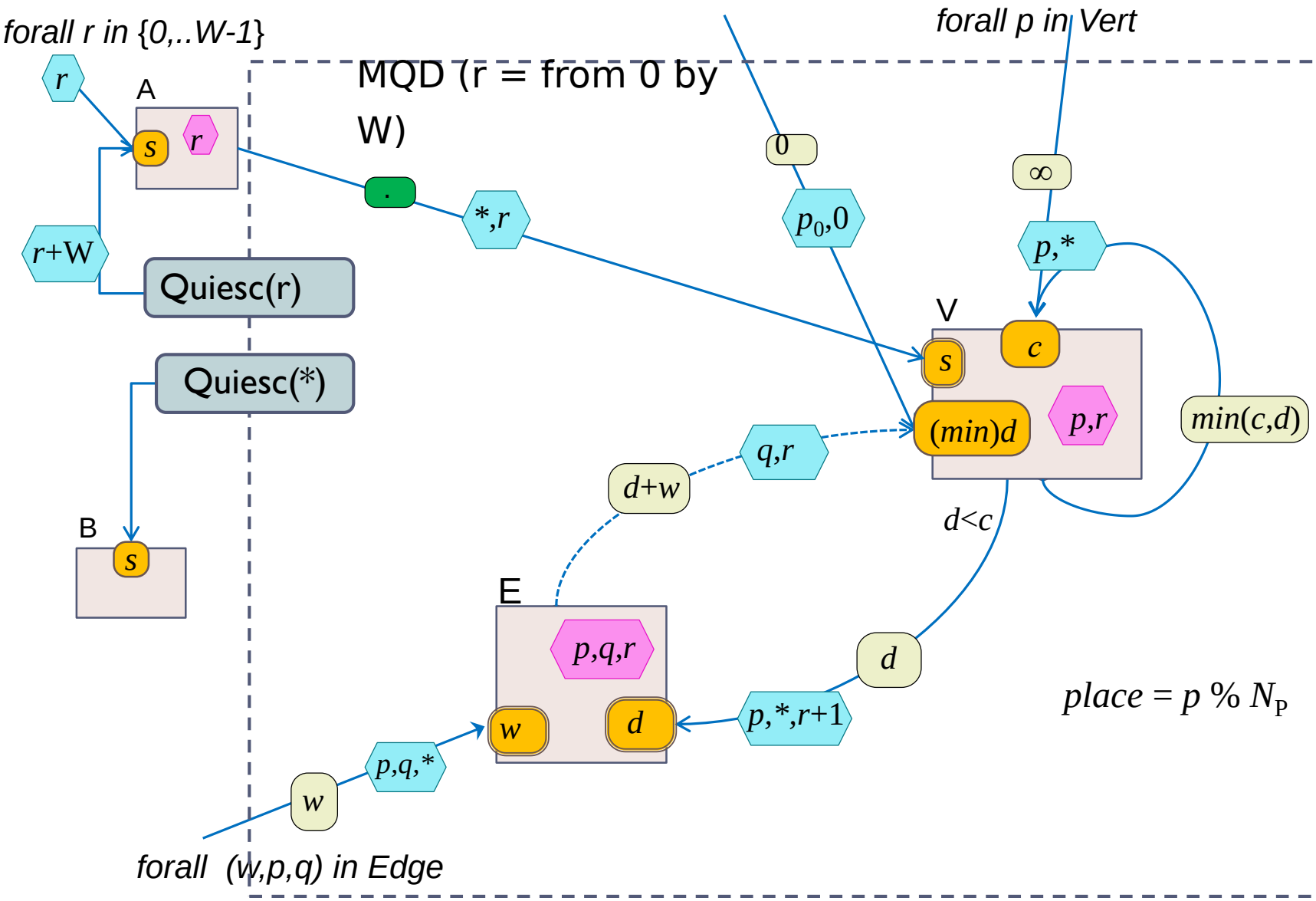
По завершению всякой активности кратчайшие расстояния от p_0 до p, будут находиться в V.c<p> . Возможны многократные проходы по дугам.
Пунктиром обозначены передачи по дугам. Только они могут быть нелокальными.
Узлы распределяются по ядрам по полю p взятием остатка от деления на число ядер N_P.

Алгоритм SSSP с синхронизацией по уровням BFS



Окружение реализует *многоуровневый детектор тишины, он* порождает токен ()⟶V.s{*r*+w,*}, когда на уровне *r* фиксируется завершение активности (w – ширина окна). В начале работы создаются w таких токенов для r=0,1…,w-1. Обычно w=1.
На входе V.d выполняется редукция через *min.*

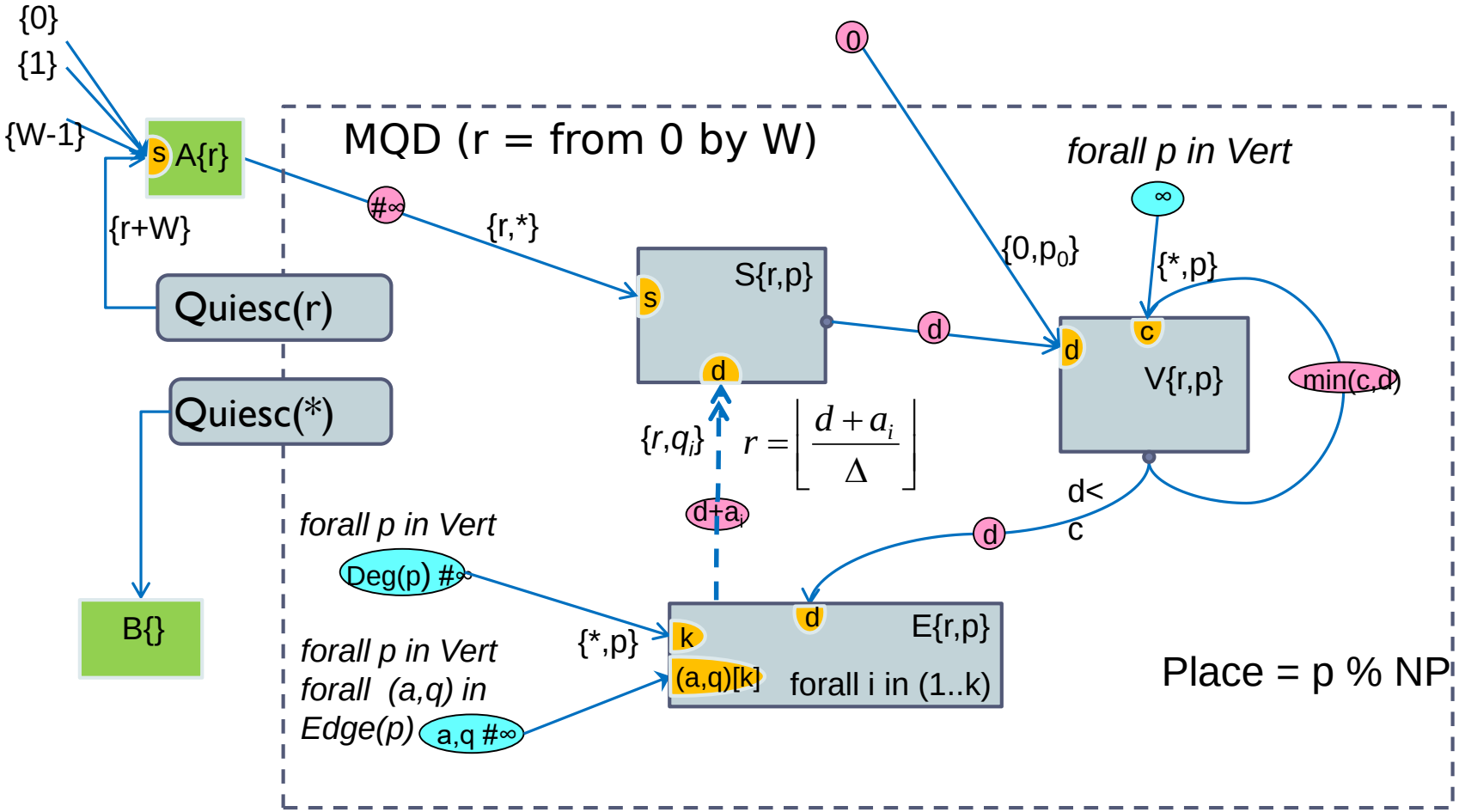
Алгоритм SSSP с синхронизацией по уровням BFS с MQD



Заключение. Проблемы и перспективы.

- Универсальнось, платформо-независимость и эффективность требуют
  - Развитую систему эквивалентных преобразований, способность их применять, проверять эквивалентность
  - Оценивать сложность (эффективность) вариантов вычислительных схем.
  - Выбирать оптимальное решение для данной платформы и данного приложения.

Схема алгоритма. Вариант с МРТ с синхронизацией по Δ-кольцам.



Многоуровневый детектор тишины активирует узел A{*r*}, когда на уровне *r* фиксируется завершение активности (w – ширина окна активности). Для стандартного варианта w=1.
В начале работы активируются первые w таких узлов для r=0,1…,w-1.
По полной тишине (во всех уровнях) активируется узел B{}.

Спасибо. Вопросы?