ИНСТИТУТ ПРОБЛЕМ ПРОЕКТИРОВАНИЯ В МИКРОЭЛЕКТРОНИКЕ РАН (ИППМ)

Архитектура потоковой модели вычислений эксафлопсного уровня

А.В. Климов,
Н.Н. Левченко, А.С. Окунев, А.Л. Стемпковский

МСКФ’2013
Москва

ПОТРЕБНОСТЬ В НОВОЙ МОДЕЛИ ВЫЧИСЛЕНИЙ

Главная проблема – обеспечение доставки данных к вычислительным устройствам.

В фон-неймановской модель вычислений возникает «стена памяти», или «узкое горло» между процессором и основной памятью.

Основное средство ее преодоления – кэш-память на чипе – создает ряд новых проблем:

- **плохая предсказуемость времени доступа,**
- **поддержка когерентности кэшей в многопроцессорных системах,**
- **сложные дополнительные механизмы управления использованием кэша.**

Модель GPGPU вводит новые виды памятей: глобальная, разделяемая, текстурная – что еще больше усугубляет проблему оптимизации перемещений данных.

Новая модель вычислений должна оцениваться прежде всего по тому, насколько легко в ней решается задача оптимальной доставки данных

ИЕРАРХИЯ ПАМЯТИ

Вид	Объем	Пропускная способность MB/s	Latency,ns
Регистры	10^2	100 000	0.4
L1	10^4	50 000	1
L2	10^5	20 000	3
L3	10^7	5 000	10
DRAM	10^{10}	5 000	100
Чужая DRAM	10^{14}	1 000	1000
Диски	10^{16}	500*	10^6

*) удельная, на 1 узел

Оптимальный внешний IO-трафик каждого уровня (для конкретной задачи) обычно убывает с ростом объема памяти данного уровня как $1/V^{\alpha}$, где $0<\alpha<1$ (α зависит от задачи).
Уровень, для которого необходима оптимизация в первую очередь, определяется параметрами иерархии памяти с одной стороны и значением α с другой стороны.

В ДОКЛАДЕ БУДУТ РАССМОТРЕНЫ

Модель вычислений с управлением данными в парадигме раздачи (УДПР) и ее свойства.

Особенности архитектуры исполняющей системы.

Пример программы в модели вычислений УДПР.

Управление пространственно-временной локализацией в модели УДПР

ОСНОВНЫЕ СВОЙСТВА МОДЕЛИ ВЫЧИСЛЕНИЙ С УПРАВЛЕНИЕМ ДАННЫМИ В ПАРАДИГМЕ РАЗДАЧИ

Активация по готовности данных:

- **Несколько значений асинхронно приходят на соответствующие входы вычислительного узла. Программа узла активируется по приходу последнего входного значения. Ей доступны только эти данные и константы (а также свой виртуальный адрес), то есть вычисление идет до конца без обращений к основной памяти или ожидания прихода других данных. Результаты посылаются как входные значения на другие узлы.**

Мелкозернистость:

- **Если типовые операции по обработке токенов и пакетов поддержаны аппаратно, становится доступным мелкозернистый параллелизм. Это позволяет свободнее выбирать порции данных/вычислений, исходя из свойств задачи, а не требований исполняющей системы.**

Виртуальная адресация узлов:

- **В отличие от традиционных систем, где данные сгруппированы в массивы (непрерывные отрезки адресуемых ячеек), в нашей модели данные привязаны к входам узлов, адресуемых векторами индексов. Виртуальная адресация узлов позволяет удобно и эффективно работать с разреженными структурами.**

Программирование в парадигме раздачи:

- **Адрес (вектор индексов) узла назначения (потребителя) определяется узлом-источником. В обычных языках реализуется парадигма сбора, когда потребитель запрашивает нужные ему данные у памяти, задавая их адреса в памяти.**

Управление локализацией

- Задается функция распределения (локализации в пространстве и времени), у которой аргументом является виртуальный адрес (вектор индексов).

Простота

- Содержательное описание алгоритма отделено от управления порядком вычислений, распределением и локальностью

НЕКОТОРЫЕ ЦИТАТЫ

Поддержка глобального адресного пространства, а не распределенной памяти.

Легкие механизмы синхронизации для быстрой передачи управления, а не принцип глобальных барьеров.

Динамические средства управления ресурсами, а не статическое распределение, выполненное при компиляции.

Обработка данных по мере их готовности, перемещая код к данным, в противоположность тому, как принято сейчас в моделях передачи сообщений.

Использование микроархитектуры, построенной на идеях обработки потока данных dataflow, а не на процессорных ядрах, которые интенсивно используют спекулятивное исполнение команд – так экономится энергия.

«Для ряда научных и исследовательских приложений, которые должны работать эффективно, надежно, хорошо масштабироваться понадобится революция в архитектуре систем. Будущие экзосистемы будут очень сильно отличаться от традиционных архитектур с распределенной памятью и передачей сообщений.»

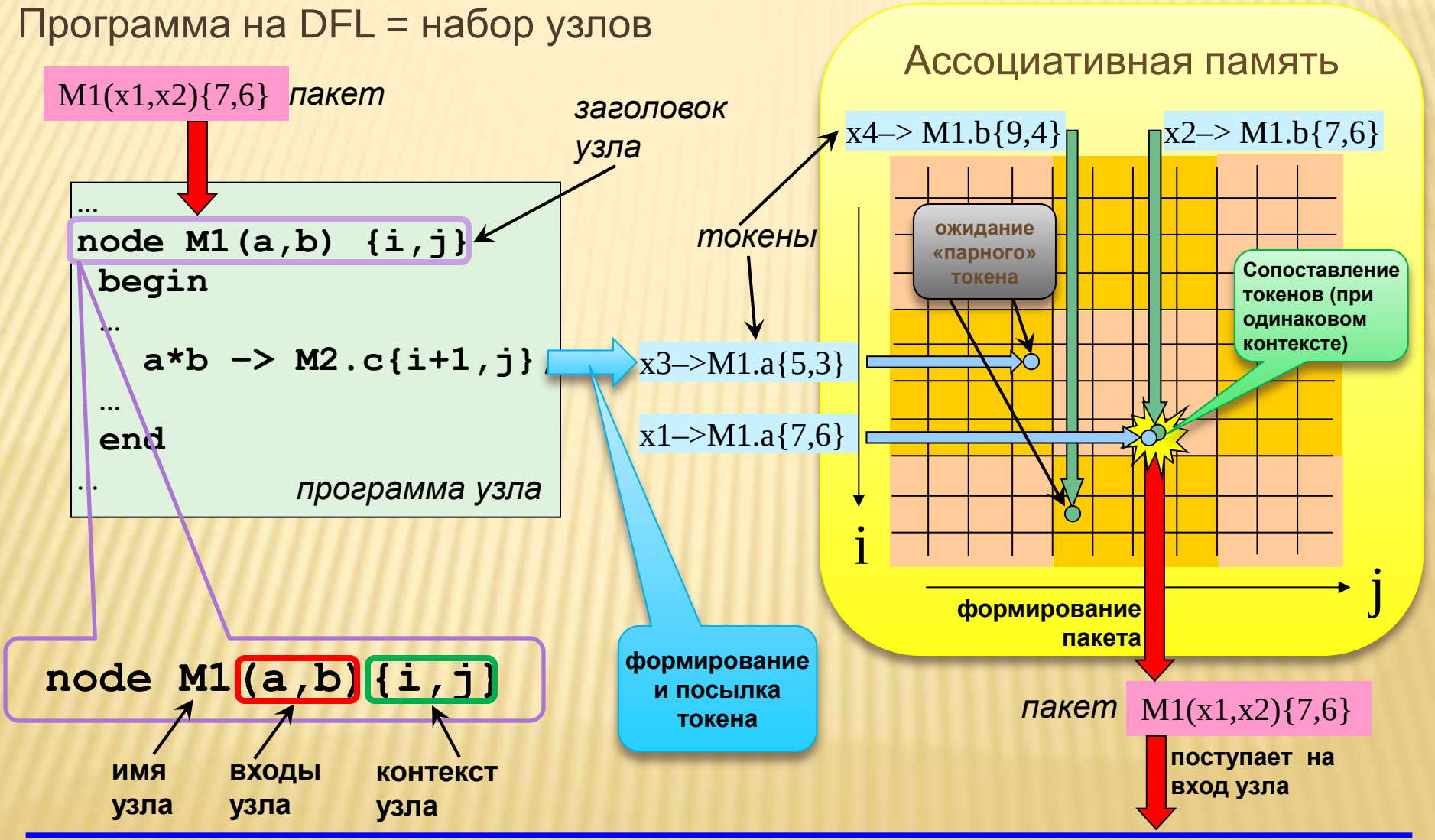
- Томас Стерлинг (профессор Университета Луизианы, создатель кластерной технологии Beowulf).

Модель dataflow позволит сделать систему не такой чувствительной к задержкам доступа к памяти и синхронизации и «выжать» максимум параллелизма из алгоритмов, потому что там можно явно прописать зависимости по данным. Я думаю, что это единственный путь для будущих систем со сверхбольшим количеством процессоров.»

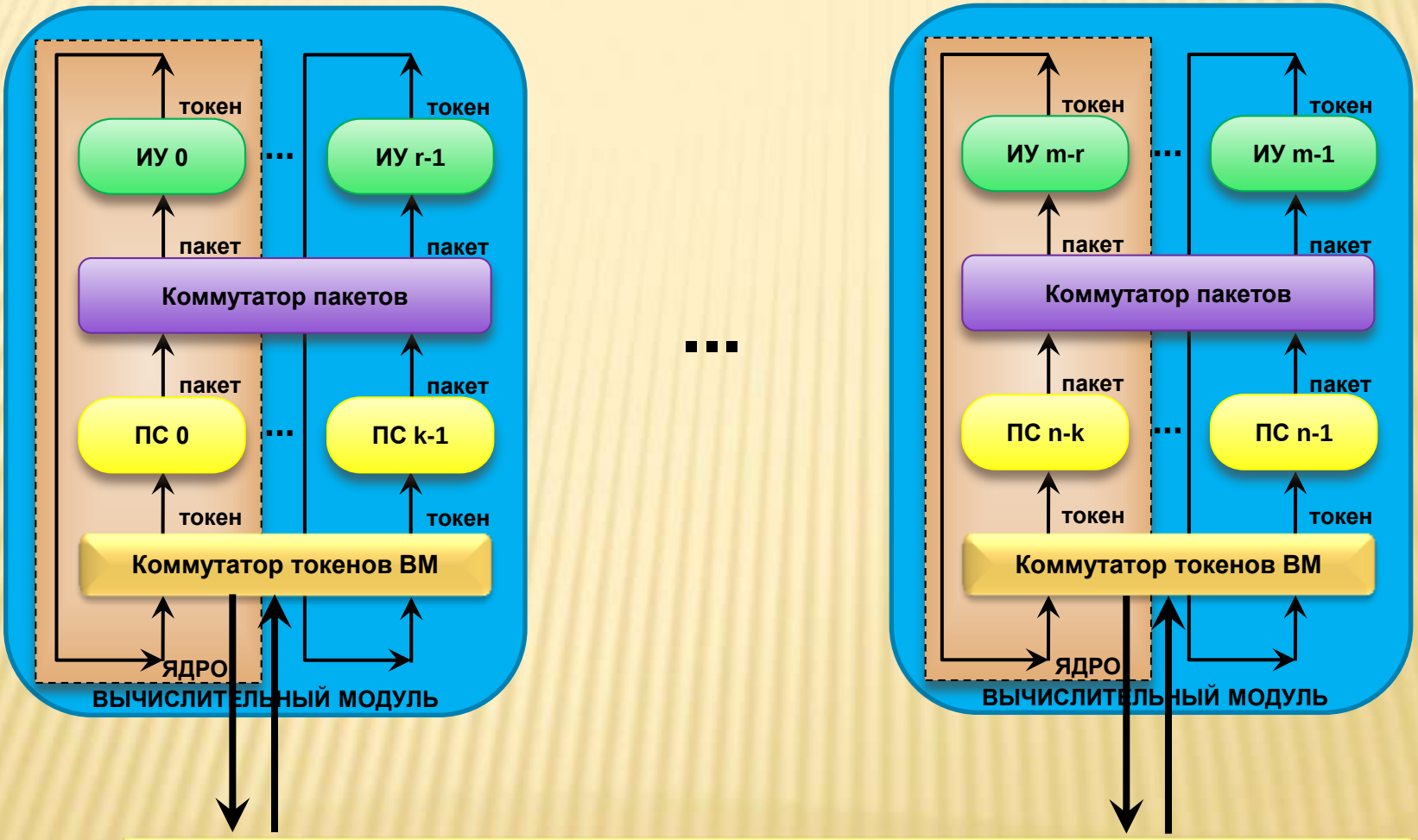
- Джек Донгарра (международный эксперт в области создания суперкомпьютеров, один из авторов списка TOP500).

МОДЕЛЬ ВЫЧИСЛЕНИЙ С УПРАВЛЕНИЕМ ПОТОКОМ ДАННЫХ В ПАРАДИГМЕ РАЗДАЧИ (УДПР)

Программа на DFL = набор узлов

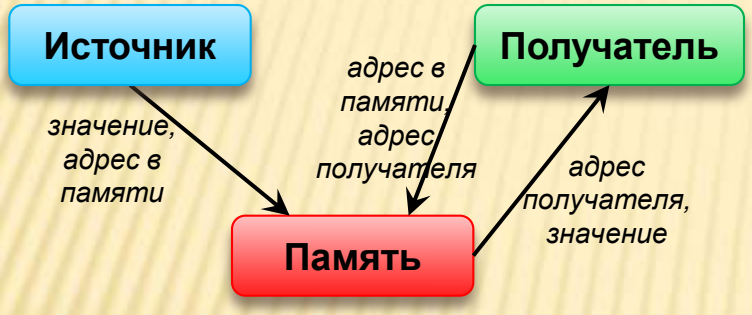


БАЗОВАЯ АРХИТЕКТУРА ПАРАЛЛЕЛЬНОЙ ПОТОКОВОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ «БУРАН»



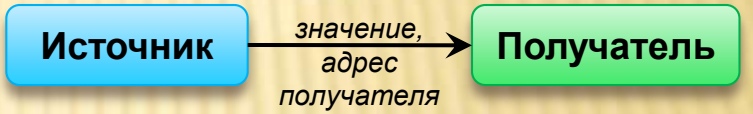
МОДЕЛЬ ВЫЧИСЛЕНИЙ УДПР ОСНОВАНА НА ПАРАДИГМЕ «РАЗДАЧИ» В ОТЛИЧИЕ ОТ ПАРАДИГМЫ «СБОРА» ТРАДИЦИОННЫХ ЯЗЫКОВ

Парадигма СБОРА
(C, Fortran, и др.)



Источник сохраняет свой результат в памяти по некоторому адресу, откуда его запрашивают потребители по мере надобности.

Парадигма РАЗДАЧИ
(DFL)



Узел-источник сам знает или вычисляет адреса всех потребителей, которым нужен его результат.

Это облегчает работу аппаратуры при распределенном выполнении.

ВОПЛОЩЕНИЕ СОВРЕМЕННЫХ ТЕНДЕНЦИЙ В АРХИТЕКТУРЕ ИСПОЛНЯЮЩЕЙ СИСТЕМЫ ППВС

МНОГОПОТОЧНОСТЬ. Каждый узел – отдельный поток. Связь между узлами – только через передачу входных данных. Потоки не приостанавливаются – только завершаются. На одном ИУ могут одновременно выполняться несколько активаций узлов, которые никоим образом не взаимодействуют между собой.

МЕЛКОЗЕРНИСТОСТЬ. Обычно стремятся увеличить размер гранулы, поскольку накладные расходы на запуск треда, прием сообщения, приостановку треда и т.п. высоки. Мы минимизируем эти накладные расходы, перекладывая их функции на аппаратуру.

АКТИВНЫЕ СООБЩЕНИЯ. Когда значения на других входах узла уже есть, токен, несущий входное значение, активирует вычисление. Поскольку любой токен может оказаться последним, каждый токен может считаться активным сообщением.

ПРОЦЕССОР В ПАМЯТИ. Токен несет адрес и данные, и он может рассматриваться как обращение к памяти, сопровождаемое запросом на обработку:

- При поступлении последнего значения на входы узла с одним контекстом (адресом) происходит активация нужной программы узла.
- Нет проблем с ожиданием завершения записи данных в память.
- Один из входов может принять адрес получателя результата.
- Возможна непосредственная обработка на входе типа редукции.

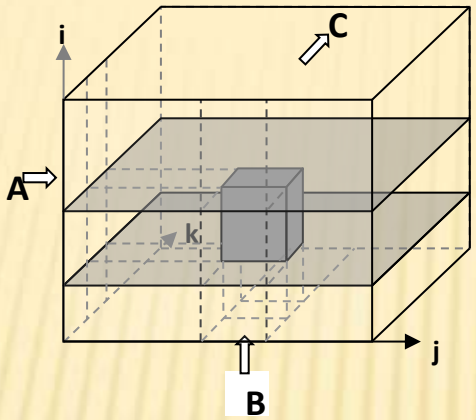
ПАМЯТЬ СОПОСТАВЛЕНИЯ (АССОЦИАТИВНАЯ). Обеспечивается функциональность памяти, аналогичная использованию full-empty bits.

DECOUPLED ACCESS-EXECUTE – DAE. Работа по принципу разделения доступа и исполнения.

ПРИМЕР: ПЕРЕМНОЖЕНИЕ МАТРИЦ

FORTRAN:

```
SUBROUTINE MM(A,B,C,N)
REAL A(N,N),B(N,N),C(N,N)
DO I = 1,N
 DO J = 1,N
  S = 0.
   DO K = 1,N
    S=S+A(I,K)*B(K,J)
   ENDDO
   C(I,J) = S
  ENDDO
 ENDDO
END
```



DFL:

```
// Inputs: A_ik → AA[i+N,k+N,  1];
//         B_kj → BB[1  ,k+N,j+N];
node AA(a)[i,k,m];
  if (m<N) a → AA[i,k,2*m],  AA[i,k,2*m+1];
  else a → M.a[i,k,m];
node BB(b)[m,k,j];
  if (m<N) b → BB[2*m,k,j], BB[2*m+1,k,j];
  else b → M.b[m,k,j];
node M(a,b) [i,k,j];
   a*b → SS.s[i,k/2,j];
node SS((+)s[2])[i,m,j];
  if (m=1) s → C[i-N,j-N] // Output: C_ij
  else s → SS[i,m/2,j];
```

PolyDFL:

```
// Inputs: A_ik → M.a[i,k,*]; B_kj → M.b[*,k,j];
node M(a,b) [i,k,j];
   a*b → S.s[i,j];
node S((+)s[n])[i,j];
   s → C[i,j];  // Output: C_ij
```

```
// Функция распределения:
place([i,k,j]) = fzip ( fnorm(i), fnorm(k), fnorm(j) );
```

УПРАВЛЕНИЕ ЛОКАЛИЗАЦИЕЙ

При использовании K процессоров (и при равномерном распределении вычислительной нагрузки) минимальный сетевой трафик для умножения матриц NxN есть

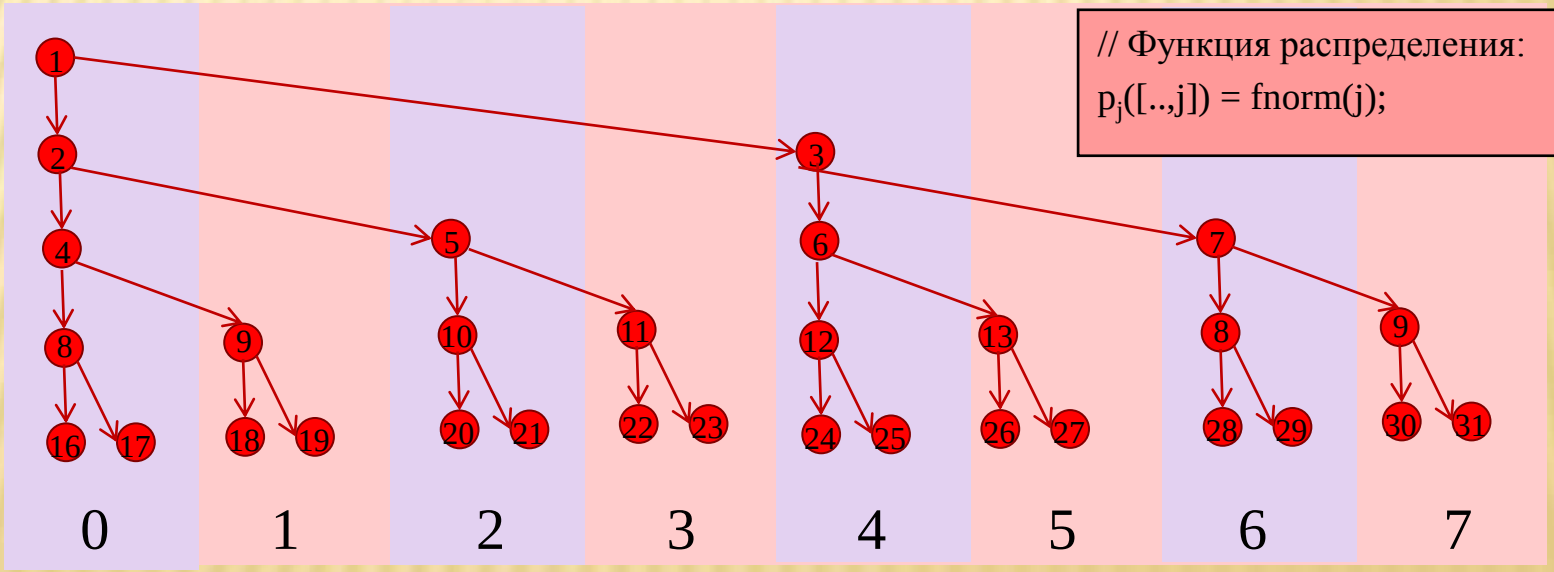
$$3N^2 \cdot \sqrt[3]{K}$$

Это значит, каждый входной элемент копируется $\sqrt[3]{K}$ раз и столько же разных процессоров дают слагаемые в один выходной элемент.

```
// Inputs: A_ik → AA[i+N,k+N,  1];
node AA(a)[i,k,m];
  if (m<N) a → AA[i,k,2*m],  AA[i,k,2*m+1];
  else a → M.a[i,k,m];
```

Копирование элемента A_{ik} вдоль измерения j размера $\sqrt[3]{K}= 8$

```
// Функция распределения:
p_j([..,j]) = fnorm(j);
```



АВТОМАТИЧЕСКАЯ ТРАНСЛЯЦИЯ (ПРИМЕР: ДВУМЕРНЫЙ МЕТОД ЯКОБИ)

FORTRAN:

```
SUBROUTINE JAC(A,L,M)
REAL A(0:L,0:L),B(0:L,0:L),X,Y
IF M.GE.1 THEN
IF L.GE.2 THEN
DO IT = 1,M
 DO J = 1,L-1
  DO I = 1,L-1
   B(I,J)=(A(I-1,J)+A(I,J-1)+
          A(I+1,J)+A(I,J+1))/4
  ENDDO
 ENDDO
 DO  J = 1, L-1
  DO I = 1, L-1
   A(I, J) =  B(I,J)
  ENDDO
 ENDDO
ENDDO
ENDIF
ENDIF
END
```

DFL:

```
module JAC;
vconst L,M: int;
type F3={it:int,j:int, i:int};

node B_1(a1:real,a2:real,a3:real, a4:real) F3{it,j,i};
    (a1+a2 +a3 + a4)/4 → A_1.b {it,j,i};

node A_1(b:real) F3 {it,j,i};
 if IT=M then
   b → A_out {i,j}
 else begin
   if J<>(L-1) then
    b → B_1.a2 {it+1,j+1,i};
   if I>=2 then
    b → B_1.a3 {it+1,j,i-1};
   if J>=2 then
    b → B_1.a4 {it+1,j-1,i};
   if I<>(L-1) then
    b → B_1.a1 {it+1,j,i+1};
 end;
endmod.
```

Автоматическая компиляция из Фортрана в DFL для программ линейного класса:

- **вложенные циклы и операторы присваивания,**
- **границы циклов и индексы массивов линейные (по параметрам циклов и константам),**
- **допускаются if с произвольным (не только линейным) условием.**

ВЫВОД

Модель вычислений УДПР обладает качествами, которые делают её конкуретноспособной в области создания и эффективного использования суперкомпьютеров эксафлопсного класса (повышается параллелизм и степень масштабируемости вычислений).

Спасибо за внимание!

Вопросы?