다중스레드 코드 생성을 위한 배열 지역화

양 창 모[†] · 유 원 희[††]

요 약

지금까지 다중스레드 모델을 위한 다중스레드 코드의 생성 및 스레드 분할에 대하여 이루어진 연구는 실행시간을 번역시간에 예측할 수 없는 연산을 스레드의 경계로 삼아 스레드를 분할하고, 스레드의 길이를 증가시키기 위하여 주어진 제약조건내에서 스레드를 병합하는 것이다. 이러한 정책으로 인하여 병렬성이 적은 프로그램이라 하여도 원격자료 접근이 많으면 스레드의 길이가 짧아지고 그에 따라 문맥전환의 수가 늘어나기 때문에 시스템에 부담이 된다.

본 논문에서는 스레드의 길이를 늘이고, 메세지 전송횟수를 감소시키기 위한 다른 방법으로 프로그램의 루프에서 접근되는 배열의 첨자를 분석하고 이를 바탕으로 루프 액티베이션에서 참조되는 배열의 원소를 해당 루프 액티베이션이 수행되는 노드에 분산 저장하는 배열의 지역화방안을 제안한다. 배열을 지역화하기 위하여 먼저 루프 액티베이션에서 접근되는 배열의 이름, 루프 첨자와 접근되는 배열원소의 첨자간의 차이인 종속거리 그리고 배열원소의 용도에 관한 정보를 얻기 위한 원소 접근 형태 분석을 수행한다. 원소 접근 형태 분석으로부터 얻어진 정보를 이용하여 가능한한 지역 기억장치에서 필요한 배열의 원소를 읽어올 수 있도록 배열 원소에 접근하는 루프 액티베이션이 수행되는 처리기 모임의 지역 기억장치에 배열 원소를 저장하는 배열 지역화를 수행한다.

실험 결과, 배열을 지역화함으로써 다른 처리기 모임의 지역 기억장치로부터 배열의 원소를 읽어오기 위한 원격자료 접근을 지역자료 접근으로 대치함으로써 원격접근의 수가 줄어든다. 이로 인하여 스레드의 길이가 증가하며, 원격접근 횟수 및 문맥 전환의 수가 줄어들어 시스템의 성능향상을 꾀할 수 있었다.

Array Localization for Multithreaded Code Generation

Changmo Yang[†] · Weohnee Yoo[††]

ABSTRACT

In recent researches on thread partitioning, partitioning algorithms break a thread at the long latency operation and merge threads to get the longer threads under the given constraints. Due to this limitation, even a program with little parallelism is partitioned into small-sized threads and context-swithings occur frequently.

In this paper, we propose another method array localization for multithreaded code generation. To localize array, we first analyze the element access patterns to extract information about the array name, dependence distance(the difference of accessed element index from loop index), and the element usage that indicates whether element is used or defined. Using this information, we can allocate array elements to the node where the corresponding loop activation is executed.

By array localization, remote accesses to array elements can be replaced with local accesses to localized array

† 정 회 원:동명전문대학 전자계산과 조교수
†† 정 회 원:인하대학교 전자계산공학과 교수
논문접수:1996년 1월 24일, 심사완료:1996년 7월 24일

elements. As a result, the boundaries of some threads are removed, programs can be partitioned into the larger threads and the number of context switchings reduced.

1. 서 론

근래의 대규모 병렬처리 시스템(massively parallel processing system)은 지역 기억장치(local memory)를 갖는 여러개의 처리기(processing element)의 모임인 노드들을 고속의 통신망으로 연결한 분산처리 시스템(distributed system)이 주종을 이루고 있다. 다중스레드 모델(multithreaded model)은 지역성을 이용한 순차코드의 효율적인 실행이 장점인 von Neumann 모델과 프로그램에 내재한 병렬성을 최대한 활용하는 데이터플로우 모델(dataflow model)은 장점을 혼합한 모델로 대규모 병렬처리 시스템을 구성하는데 적당한 실행 모델로 근래 많은 연구가 이루어지고 있다. 다중스레드 모델에서 프로그램은 함수, 루프 등이 단위가 되는 코드블럭(code block)으로 구성되며, 각 코드블럭은 번역시간에 실행순서가 결정될 수 있는 명령의 모임인 스레드로 분할된다. 다중스레드 모델은 실행 가능한 스레드들간의 문맥전환(context switching)을 통하여 동기화(synchronization)나 원격자료 접근(remote data access)등에서 발생하는 지연(latency)을 은폐함으로서 처리기의 활용도를 향상시킨다[1, 5, 6, 7, 9, 10, 11, 14, 15, 16].

이러한 다중스레드 모델을 위한 코드생성의 기본 원칙으로 병렬성의 극대화, 스레드 길이의 최대화, 동기화의 최소화, 교착상태 회피, 자원활용도의 극대화 등이 있다[9]. 지금까지 다중스레드 모델을 위한 다중스레드 코드의 생성 및 스레드 분할에 대하여 이루어진 연구는 주로 순차코드의 모임인 스레드의 길이를 향상시키기 위하여 주어진 제약조건내에서 스레드를 병합하는 것이다[4, 8, 9, 17, 18].

기존의 스레드 분할 방법은 동일한 입력이나 출력을 갖는 연산들을 하나의 스레드로 묶으면서, 원격접근과 동기화로 인한 지연시간을 은폐하기 위하여 함수 호출, 루프 호출, 원격자료 접근 등 실행시간이 번역시간에 정의되지 못하는 연산을 경계로 삼는다. 이로 인하여 병렬성이 적은 프로그램이라 하여도 원격자료 접근이 많으면 스레드의 길이가 짧아지고 그에 따라 문맥전환의 수가 늘어나기 때문에 시스템에 부담이 된다.

분산처리 시스템에서 또 한가지 고려할 사항은 참조의 지역성(locality of reference)이다. 분산처리 시스템은 다른 처리기의 지역 기억장소에 있는 자료인 원격자료에 접근하기 위하여 메시지 전송(message passing) 방법을 사용한다. 이 메시지 전송은 지역 기억장소에 접근하는 것보다 비용이 훨씬 더 많이 들기 때문에 가능한 한 지역자료에 접근할 수 있도록 자료를 처리기사이에 분산시켜야 한다[12, 13, 19].

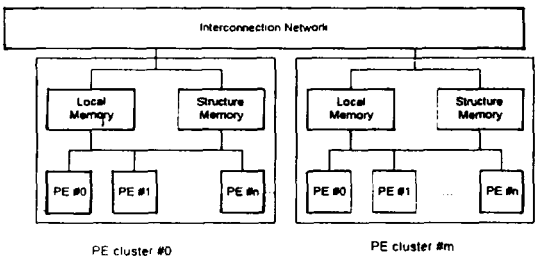
본 논문에서는 스레드의 길이를 늘이고, 원격접근 횟수를 감소시키기 위한 다른 방법으로 배열의 지역화방안을 제안한다. 배열지역화는 프로그램의 루프 액티베이션(activation)에서 참조되는 배열의 첨자를 분석하여 해당 루프 액티베이션에서 접근되는 배열의 원소를 루프 액티베이션이 수행되는 노드에 분산 저장한다. 배열을 지역화하면 루프 액티베이션에서 접근하는 자료가 그 루프 액티베이션을 수행하는 노드의 지역 기억장치에 존재하므로, 원격자료 접근의 수를 감소되어 원격자료 접근으로 인한 통신량이 감소되며, 스레드의 길이를 증가시켜 문맥전환의 수를 줄임으로서 시스템의 성능향상을 꾀할 수 있다.

본 논문에서 사용하는 다중스레드 모델은 기억장치를 공유하는 처리기들로 구성된 처리기 모임(PE cluster)의 모임이며, 사용언어는 I-구조(I-structure)[2, 3]를 갖는 함수언어로 I-배열을 배열지역화의 대상으로 삼는다.

2. 다중스레드 모델

다중스레드 모델에서 프로그램은 여러개의 코드블럭으로 구성되는데 여기서 코드블럭은 한 문맥(context)을 나타내며, 보통 함수나 루프 몸체에 해당된다. 코드블럭은 관련된 여러개의 스레드로 구성되며, 스레드는 실행 순서가 번역시간에 결정된 명령어의 모임으로 정의된다. 기존의 다중스레드 모델에서 배열은 한 노드의 지역 기억장치에 할당되어 다른 노드의 처리기가 배열 원소에 접근하려면 원격 기억장치 접근을 해야한다. 다중스레드 모델에서 원격 기억장치 참

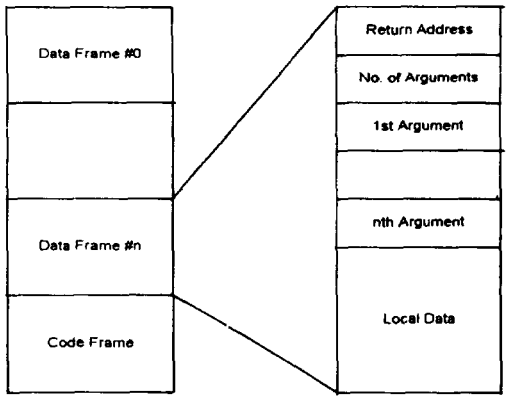
조와 같이 긴 지연시간을 초래하는 명령어는 분할수행(split transaction)되며, 스레드의 경계가 된다. 이로써 원격 기억장치에 있는 자료에 대한 요청 발생시 처리기는 쉬지 않고 다른 명령어를 계속 수행하거나 다른 스레드로 실행을 전환할 수 있다. 분할 수행을 위해서 원격 요청에 대한 발신자와 수신자는 서로 다른 스레드에 속하게 된다.



(그림 1) 다중스레드 기계의 추상 구조
(Fig. 1) Multithreaded Architecture

본 논문에서 사용하는 다중스레드 모델은 그림 1과 같이 상호연결망(interconnection network)에 연결된 처리기 모임의 모임으로 구성된다. 처리기 모임은 하나의 지역 기억장치와 구조 기억장치(structure memory)를 공유하는 다수의 처리기로 이루어진다. 각 처리기는 함수, 루프 등의 활성화된 코드블럭(activated code block)을 수행하며, 지역 기억장치는 코드블럭을 수행하는데 필요한 코드와 자료를 보관하는 코드프레임(code frame)과 자료프레임(data frame)을 저장하고 구조 기억장치에는 해당 처리기 모임이 사용하는 배열을 저장한다. 새로운 코드블럭이 활성화될 때마다 자료프레임이 지역 기억장치에 생성되고, 하나의 처리기에 할당되어 수행을 시작한다. 다중스레드 모델의 지역 기억장치와 자료프레임의 구조는 그림 2와 같다.

다중스레드 모델에서의 루프는 순차루프(sequential loop)와 병렬루프(parallel loop)로 나뉘어진다. 순차루프는 루프 액티베이션간에 종속이 존재하여 전 액티베이션이 완전히 종료한 후에 다음 액티베이션이 수행될 수 있는 루프로 하나의 자료프레임을 순차루프의 모든 액티베이션이 공용한다. 병렬루프는 루프



(그림 2) 지역기억장치와 자료 프레임
(Fig. 2) Local Memory and Data Frame

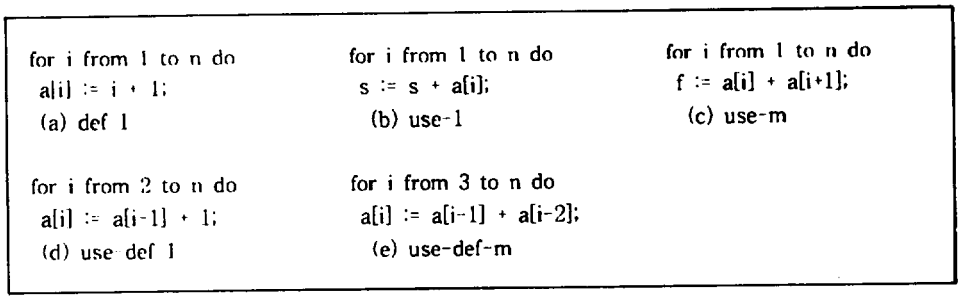
액티베이션간에 종속이 존재하지 않으므로 각 액티베이션이 병렬적으로 수행할 수 있다. 따라서 병렬성을 극대화하기 위하여 병렬루프가 호출되면 지정된 수만큼의 자료프레임이 할당되고, 루프를 실행하는데 필요한 값들이 도착하면 바로 실행을 시작한다.

루프를 수행하는 과정은 루프 설정, 루프 실행, 루프 종료의 세과정으로 구성된다. 순차루프의 설정은 루프가 호출되면 그 루프의 자료프레임을 지역 기억장치에 할당하고 루프의 첨자 등 루프를 수행하는데 필요한 값이 도착하기를 기다린다.

값이 모두 도착하면 한 루프 액티베이션이 실행되고, 루프 액티베이션이 실행되면서 얻어진 값들을 자료프레임이 적당한 위치에 저장한다. 한 루프 액티베이션이 종료되면 다음 액티베이션이 전 액티베이션이 사용하던 자료프레임을 재사용하므로 약간의 병렬성은 손실되더라도 루프 액티베이션간의 자료 이동은 존재하지 않아 자료 전송의 부담이 없어진다. 루프 종료는 루프의 결과를 루프 호출자에게 돌려주고, 루프가 사용한 자료프레임을 실행시간 시스템에게 반환하는 과정이다. 병렬루프의 수행은 순차루프와 비슷하지만 액티베이션들이 동일한 자료프레임을 사용하는 것이 아니라 별도의 자료프레임을 할당받아 병렬적으로 수행된다는 점이 다르다.

3. 배열 접근 형태 분석

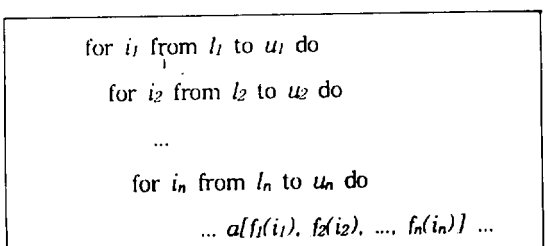
본 논문의 다중스레드 모델은 루프의 한 반복(iter-



(그림 3) 배열 접근 형태의 예
(Fig. 3) Examples of Array Access Patterns

ation)이나 함수호출을 하나의 액티베이션으로 취급하여 하나의 자료프레임을 할당하고 한 처리기 모임에서 실행한다. 배열지역화를 행하려면 먼저 배열 원소의 접근 형태를 분석하여야 한다. 이 장에서는 루프 내에서 배열의 원소가 접근되는 형태를 분석하기 위한 배열 종속 분석을 설명한다.

배열의 원소가 하나의 액티베이션에서 접근되는 형태는 다음 5가지로 나눌 수 있다. 첫째, 배열의 원소가 한 액티베이션에서 정의만 되는 형태(def-1), 둘째, 배열의 원소가 한 액티베이션에서 사용만 되는 경우(use-1), 셋째, 배열의 원소가 여러 액티베이션에서 사용되는 경우(use-m), 넷째, 배열의 원소가 한 액티베이션에서 사용되고 다른 한 액티베이션에서 정의되는 경우(use-def-1), 다섯째, 배열의 원소가 한 액티베이션에서 정의되고, 다른 여러 액티베이션에서 사용되는 경우이다(use-def-m). I-구조의 의미상 배열의 한 원소가 여러 액티베이션에서 정의되는 경우는 없다. 1차원 배열 및 반복만을 고려한다면 그림 3이 각 경우의 예이다.



(그림 4) 일반적인 다차원 루프
(Fig. 4) Multimensional Loop

그림 4는 n차원 배열 $a[1:u_1, 1:u_2,..., 1:u_n]$의 원소에 접근하는 n차원 루프이고, 그림 5는 그림 4와 같은 루프에서의 원소 접근 형태를 분석하는 알고리즘이다. 이 알고리즘은 원시 프로그램을 입력받아 배열 원소의 접근 형태와 한 루프 액티베이션에서 접근하는 원소를 추출함으로서 배열지역화를 수행하기 위한 정보를 얻어낸다.

원소 접근 형태 분석에서 얻어지는 정보는 접근되는 배열의 이름, 루프 첨자로부터 원소 첨자로의 종속거리(dependence distance), 그리고 원소의 값을 사용하는가 또는 정의하는가를 나타내는 용도(usage)로 구성된 triple의 집합 S이다. 이 정보는 후에 배열의 지역화에 사용되는데 이때 가장 중요한 것이 루프 첨자와 원소 첨자의 종속거리이다. 종속거리는 루프첨자에 대한 함수로 해당 루프 액티베이션이 수행되는 처리기 모임의 지역 기억장치에 저장될 배열의 원소 첨자를 계산하는데 사용된다.

원소 $a[f_1(i_1), f_2(i_2),..., f_n(i_n)]$의 종속거리 d는 배열의 각 첨자 $i_j(1 \le j \le n)$에 대하여 $\sum_{j=1}^{n} d_j(\prod_{k=j+1}^{n} u_k)$이다. 이때 $d_j = f_j(i_j) - i_j$이다. 이 정보를 얻은 후에 배열의 이름과 용도가 같은 원소를 모아 두 집합 T_1과 T_2에 저장한다. T_1은 배열 원소의 값을 사용하는 접근의 집합이고 T_2는 배열 원소의 값을 정의하는 접근의 집합이다. 배열 원소의 접근 형태는 집합 T_1과 T_2의 농도(cardinality)에 따라 결정된다. 그림 3의 (e)의 경우 각 원소 접근 $a[i]$, $a[i-1]$, $a[i-2]$에 대한 종속거리는 각각 0, −1, −2이고, 그림 6과 같은 wavefront 프로그램의 이차원 루프에서 배열 접근 $m[i-1, j]$, $m[i, j-1]$, $m[i-1, j-1]$의 종속 거리는 각각 $-u$,

```
알고리즘 배열 원소 접근 형태 분석
입력    원시프로그램
출력    배열 a의 접근형태 a.p와 종속거리 a.d
변수    배열 원소 접근 정보 집합 S, T1, T2
        배열 원소 접근 정보 집합은 triple (aname, dist, usage)의 집합이다.
        aname은 배열명이다.
        dist는 배열원소의 종속거리이다.
        용도 usage는 배열원소의 값을 사용하는 접근이면 U,
                    정의하는 접근이면 D이다.
방법
    S = ∅;
    for 모든 루프내 배열 원소 접근 b[e]에 대하여 do
      종속거리 dist를 계산한다.
      b[e]가 식의 좌변에 나오면 usage는 D, 우변에 나오면 U이다.
      S = S ∪ {(b, dist, usage)}
    endfor
    while S ≠ ∅ do
      T1 = ∅; T2 = ∅;
      S에서 임의의 원소 s1 = (a e, U)를 선택한다.
      T1 = T1 ∪ {s1}; S = S - {s1};
      S에서 이름이 a이고 usage가 U인 모든 원소 si를 선택한다.
      T1 = T1 ∪ {si}; S = S - {si};
      S에서 임의의 원소 s2 = (a e, D)를 선택한다.
      T2 = T2 ∪ {s2}; S = S - {s2};
      S에서 이름이 a이고 usage가 D인 모든 원소 sj를 선택한다.
      T2 = T2 ∪ {sj}; S = S - {sj};
      if |T1| = 0 and |T2| = 1
          then a.p = def-1;
               a.d = T2의 원소의 종속거리 endif;
      if |T1| = 1 and |T2| = 0
          then a.p = use-1;
               a.d = T1의 원소의 종속거리 endif;
      if |T1| > 1 and |T2| = 0
          then a.p = use-m;
               a.dn = T1의 원소에서 최소 종속거리;
               a.dx = T1의 원소에서 최대 종속거리 endif;
      if |T1| = 1 and |T2| = 1
          then a.p = use-def-1;
               a.d = T1의 원소의 종속거리 endif;
      if |T1| >1 and |T2| = 1
          then a.p = use-def-m;
               a.dn = T1의 원소에서 최소 종속거리;
               a.dx = T1의 원소에서 최대 종속거리 end-if;
    end-while;
```

(그림 5) 배열 원소 접근 형태 분석 알고리즘
(Fig. 5) Array Access Pattern Analysis Algorithm

```
for i from 2 to u do
  for j from 2 to u do
    m[i, j] = (m[i-1, j] + m[i, j-1] + m[i-1, j-1]) / 3;
```

(그림 6) Wavefront 프로그램내의 이차원루프
(Fig. 6) Two dimensional Loop in Wavefront

-1, $-u-1$이다.

4. 배열지역화

이 장에서는 3장에서 얻어진 배열 원소 접근 형태와 종속거리를 이용하여 배열을 처리기 모임의 지역 기억장치에 분산, 할당하는 정책을 설명한다.

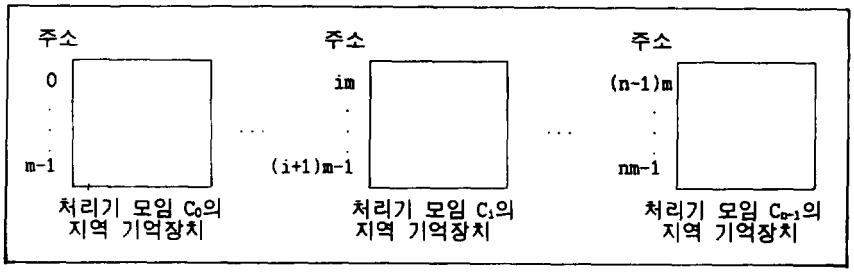
배열 원소 접근 형태가 def-1 또는 use-1이면 배열 원소의 접근이 한 액티베이션에서만 이루어지므로 그 원소를 해당 액티베이션이 사용하는 자료프레임에

저장하여 지역자료로 사용할 수 있다. use-m의 경우 하나이상의 액티베이션에서 배열원소에 접근하지만 읽기 연산만을 하므로 자료가 복사되어도 참조 명료성(referential transparency)을 해치지 않는다. 따라서 배열 전체를 액티베이션이 수행되는 노드의 지역 기억장치마다 복사하거나 각 액티베이션에서 참조하는 원소만을 복사한다. 전자의 경우, 기억공간이 낭비되지만 배열 할당 및 원소의 주소를 계산하는 방법이 간단한 반면 후자의 경우는 반대이다.

배열 원소 접근 형태가 use-def-1의 경우는 배열 원소에 접근하는 액티베이션(그림 3의 (d)에서 i번째 액티베이션과 i-1번째 액티베이션)간에 종속이 존재함을 의미하므로 최소한 한 액티베이션에서는 원격접근을 해야한다. I-구조를 사용하는 다중스레드 모델에서 원격접근 연산중 배열의 원소 값을 읽어오는 I-fetch 연산은 원격접근의 결과를 기다려야 하므로 분할 수행되어 스레드의 경계가 되지만 원소의 값을 저장하는 I-store 연산은 원소의 값을 저장하라는 메시지 전송 후에 결과를 기다릴 필요가 없으므로 I-store 연산은 스레드 분할의 경계가 되지 않는다. 이러한 사실로부터 그림 3의 (d)와 같은 use-def-1의 경우 원소 $a[i]$를 i번째 액티베이션의 지역자료로 저장한다. 같은 맥락에서 그림 3의 (e)와 같은 use-def-m의 경우 i번째 액티베이션은 원소 $a[i-1]$과 $a[i-2]$를 지역자료로 저장하여 이에 대한 I-fetch 연산은 지역접근으로 하고, $a[i]$를 저장하는 I-store 연산은 원격접근을 하도록 한다.

본 논문에서 사용하는 다중스레드 모델은 그림 7과 같이 지역 기억장치가 모여 전역주소공간(global address space)을 구성한다. 각 지역 기억장치에는 처리기 모임에 속한 처리기의 수만큼 액티베이션이 할당되며, 각 액티베이션에서 사용하는 배열을 저장한다. 각 지역 기억장치의 크기를 m, 시스템내 처리기 모임의 수를 n이라 할 때 처리기 모임 $C_i(0 \le i \le n-1)$의 지역 기억장치를 주소 범위는 $i \cdot m$번지에서 $(i+1) \cdot m - 1$까지이다. 한 번지에는 하나의 값이 저장될 수 있다고 하자. 본 논문에서는 다차원 배열을 선형화(linearization)하여 차원의 구분없이 원소의 순서에 따라 처리기 모임의 지역 기억장치에 할당한다.

처리기 모임내 처리기의 수를 p라 하고 첨자 시작값이 s(그림 6의 경우 u+2), 종료 값이 l인 루프에서 종속거리가 d인 배열 $a[1:u_1, 1:u_2, \ldots, 1:u_n]$의 원소를 참조한다고 하자. 이때 참조 형태에 따라 배열 a는 다음과 같이 지역 기억장치에 할당된다. 먼저 참조 형태가 use-m 또는 use-def-m이라면 한 액티베이션에서 사용되는 배열 원소가 2개이상이다. 이 경우에는 두가지의 배열할당을 고려할 수 있다. 하나는 해당 액티베이션이 사용하는 둘 이상의 원소를 지역 기억장소에 저장하는 완전 지역화(full localization)이고, 다른 하나는 사용되는 원소중 하나만을 지역 기억장소에 저장하는 부분 지역화(partial localization)이다. 첫 번째 루프 액티베이션이 처리기 모임 C_q에서 수행된다할 때, 부분 지역화를 적용하는 경우, 처리기 모임 C_i의 지역 기억장치에는 $s + d_x + (i-q) \cdot p$번째 원소에서 $s + d_x + (i-q+1) \cdot p - 1$번째 원소를 저장하고, 완전 지역화를 적용하는 경우에는 $s + d_x + (i-q) \cdot p$번째 원소에서 $s + d_x + (i-q+1) \cdot p - 1$번째 원소를 저장한다. 이때 배열원소 a의 k번째 원소의 전역주소는 $(q + k \ div \ p) \cdot m + k \ mod \ p$번지이다. 여기서 div는 정수 나눗셈, mod는 나머지 연산자이다. 이때



(그림 7) 전역 주소 공간
(Fig. 7) Global Address Space

한가지 고려할 사항은 배열 원소 접근 형태가 use-def-m의 경우이다. 이 경우에는 배열의 원소가 복사되어 여러개의 처리기 모임에 분산되었을 때 배열의 원소가 정의되어 그 값을 저장하기 위하여 여러개의 처리기 모임에 전달하는 경우 통신량이 증가할 것이다. 따라서 사용되는 원소중 임의의 원소를 골라 다음에 설명할 use-1이나 def-1의 경우처럼 배열을 분산할 수도 있다.

use-1 또는 def-1의 경우 첨자 거리를 d라 하면, 처리기 모임 C_i의 지역 기억장치에는 $s+d+(i-q)\cdot p$번째 원소에서 $s+d+(i-q+1)\cdot p-1$번째 원소를 저장한다. 이때 배열 a의 k번째 원소의 전역주소는 $(q+k\ div\ p)\cdot m+k\ mod\ p$번지이다. def-1의 경우는 배열분산을 단순하게 하기 위하여 배열을 선언한 액티베이션에 배열 전체를 할당한다.

4개의 액티베이션이 한 처리기 모임에서 수행되고, 그림 6에서 u가 8이라면, 표 1과 같이 배열이 지역화된다.

〈표 1〉 배열지역화의 예
〈Table 1〉 Example of Array Localization

처리요소 모임 번호	첨자(i, j)	저장되는 원소	
		완전 지역화	부분 지역화
0	(2, 2)-(2, 5)	(1, 1)-(2, 4)	(1, 1)-(1, 4)
1	(2, 6)-(3, 2)	(1, 5)-(3, 1)	(1, 5)-(1, 8)
2	(3, 3)-(3, 6)	(2, 2)-(3, 5)	(2, 2)-(2, 5)
3	(3, 7)-(4, 3)	(2, 6)-(4, 2)	(2, 6)-(3, 2)
4	(4, 4)-(4, 7)	(3, 3)-(4, 6)	(3, 3)-(3, 6)
5	(4, 8)-(5, 4)	(3, 7)-(5, 3)	(3, 7)-(4, 3)
6	(5, 5)-(5, 8)	(4, 4)-(5, 7)	(4, 4(-(4, 7)
7	(6, 2)-(6, 5)	(5, 1)-(6, 4)	(5, 1)-(5, 4)
8	(6, 6)-(7, 2)	(5, 5)-(7, 2)	(5, 5)-(6, 1)
9	(7, 3)-(7, 6)	(6, 2)-(7, 5)	(6, 2)-(6, 5)
10	(7, 7)-(8, 3)	(6, 6)-(8, 4)	(6, 6)-(7, 2)
11	(8, 4)-(8, 7)	(7, 3)-(8, 6)	(7, 3)-(7, 6)
12	(8, 8)	(7, 7)-(8, 7)	(7, 7)-(8, 3)

5. 실 험

본 논문에서 제안한 배열지역화 방법으로 인한 스레드 크기의 향상과 원격접근 횟수의 감소등을 보이기 위하여 스레드 분할시 배열지역화를 적용한 경우와 적용하지 않은 경우 그리고 부분적으로 지역화한 경우에 대하여 스레드의 전환횟수와 실행되는 명령어의 수 및 원격 접근의 횟수를 배열 원소의 수를 변화시키면서 비교하였다. 실험대상은 fibo(피보나치수열), loop3(Lawrence Livermore Loop 3), MA(Matrix Addition), wave(Wavefront)이다. fibo와 wave는 반복간에 종속관계가 존재하는 순차루프이며, MA와 loop3는 병렬루프이다. 각 그래프에서 NL(No Localization)은 지역화를 하지 않은 경우, FL(Full LOcalization)은 모든 배열을 지역화한 경우이고, PL(Partial Localization)은 부분적으로 지역화를 한 것이다.

실행명령의 수를 비교하는 그래프에서 지역화를 하지 않는 경우에 비하여 지역화를 한 경우의 명령수가 적은 것은 지역화를 할 경우 원격접근대신 지역접근이 가능하므로 스레드 내에서 지역성을 유지할 수 있기 때문이다. 원격접근이 사용된다면 원격접근을 요청하는 위치에서 원격접근의 결과는 다른 스레드에서 사용하므로 스레드 내에서 지역성을 활용할 수 있는 지역접근에 비하여 원격접근의 결과를 읽어오는 연산이 추가로 필요하므로 명령의 수가 늘어난다.

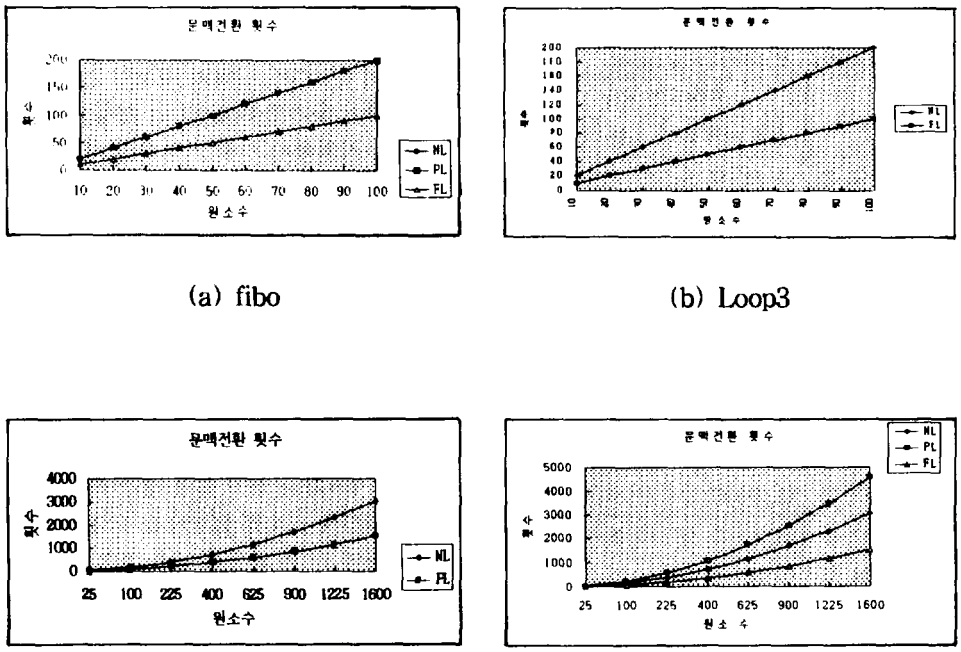
완전 지역화와 부분 지역화의 차이점은 원격 접근의 종류이다. 완전 지역화를 하였을 때 발생하는 원격 접근은 중복되어 저장된 배열 원소의 값을 정의했을 때 다른 곳에 저장된 배열 원소의 값도 같이 정의하여야 하기 때문에 발생한다. 부분 지역화를 하였을 때 발생하는 원격 접근은 물론 지역화가 되지 않은 배열 원소에 접근하기 위하여 발생한다.

루프간 종속이 존재하는 fibo와 루프간 종속이 존재하지 않는 loop3을 예로서 실험결과를 분석한다. fibo는 지역화를 하지 않은 경우 루프 몸체는 i-1번째 또는 i-2번째 원소를 읽어오는 스레드와 두 원소의 값을 더하여 i번째 원소의 값을 정의하는 스레드 등 두 개의 스레드로 분할되며, 한 루프 액티베이션당 i번째 원소, i-1번째 원소 그리고 i-2번째 원소에 접근하여야 하므로 세번의 원격 접근이 발생하고, 문맥전환은 두 번 발생한다. 완전 지역화를 한 경우 루프 몸체는 모든 원격 접근이 지역 접근으로 변환되므로 스레드의 경계가 없어져 하나의 스레드로 분할된다. i번째 원소가 중복저장되므로 한 루프 액티베이션당 한번의 원

격 접근이 발생하고, 문맥전환은 한번 발생한다. i-1번째 또는 i-2번째 원소중 하나의 원소만을 지역화(부분 지역화)한 경우 루프 몸체는 두개의 스레드로 분할되며, 한 루프 액티베이션당 두번의 원격 접근, 두번의 문맥전환이 발생한다.

loop3은 루프간 종속이 존재하지 않기 때문에 한 루프 액티베이션에서 접근되는 배열 원소는 다른 루프 액티베이션에서는 접근되지 않는다. 따라서 완전 지역화만이 가능하다. 지역화를 하지 않았을 경우 loop3의 루프 몸체는 두개의 스레드로 분할되어 두번의 문맥전환이 발생하고, 세번의 원격 접근이 필요하다. 지역화를 했을 경우 루프 몸체는 하나의 스레드로 분할되어 한 루프 액티베이션이 수행될 때 한번의 문맥전환이 발생하고 원격 접근은 발생하지 않는다.

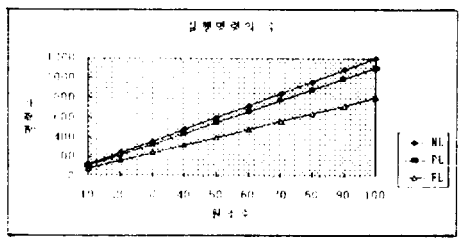
이상의 실험 결과로 보아 부분 지역화 혹은 완전 지역화를 하는 것이 지역화를 하지 않는 것보다 실행되는 명령의 수, 문맥전환 횟수, 원격 접근 횟수가 모두 줄어듬을 알 수 있다. 다만 한가지 고려할 사항은 순차루프의 경우 부분 지역화와 완전 지역화중 어느 것을 택할 것인가하는 것이다. 만일 접근되는 원소의 첨자와 루프 첨자간의 차인 종속거리가 먼 경우 완전 지역화를 하게되면 중복 저장되는 원소의 수가 많아질 수 있으며 이로 인하여 중복된 원소 값을 저장하기 위한 원격 접근의 횟수가 루프내에서 접근되는 원소의 수보다 많아지는 경우는 부분 지역화가 완전 지역화보다 적당할 것이다. 이러한 상황은 주로 이차원 이상의 배열에서 루프간 종속이 원소단위가 아닌 면 또는 행에 존재하는 경우에 발생한다. 이를 판단하여 적당한 경계를 찾는 것도 추가로 연구되어야 할 사항이라 생각된다.



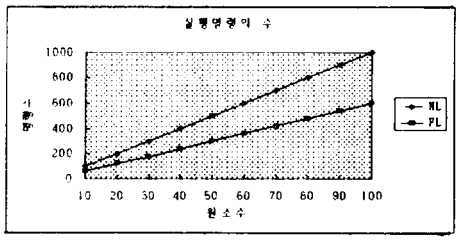
(a) fibo (b) Loop3

(c) MA (d) Wave

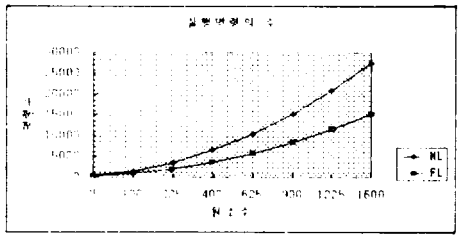
(그림 8) 문맥전환 횟수 비교 그래프
(Fig. 8) Comparison of number of context switchings



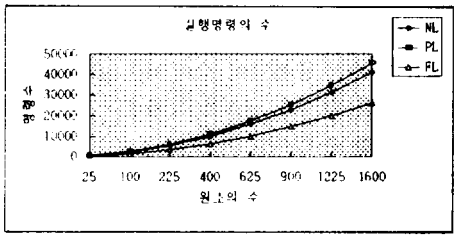
(a) fibo



(b) Loop3

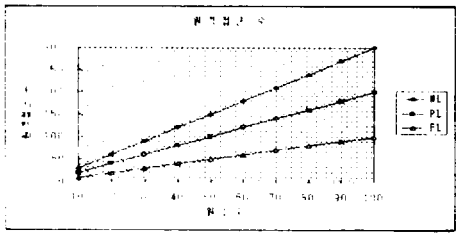


(c) MA

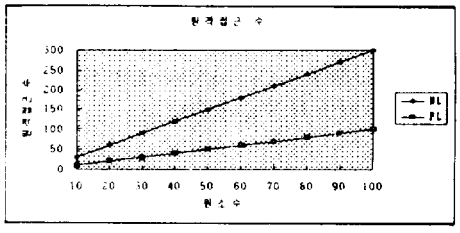


(d) Wave

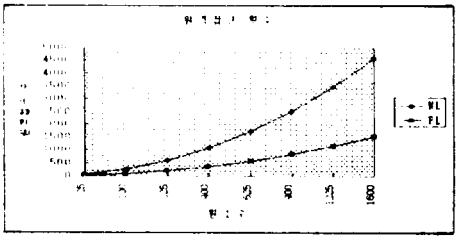
(그림 9) 실행명령의 수 비교 그래프
(Fig. 9) Comparson of number of executed instructions



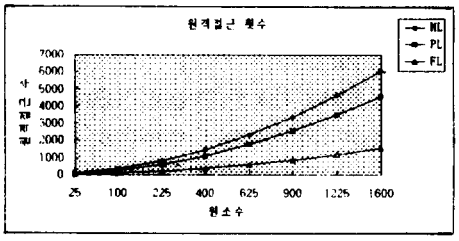
(a) fibo



(b) Loop3



(c) MA



(d) Wave

(그림 10) 원격접근 횟수 비교 그래프
(Fig. 10) Comparison of number of remote accesses

6. 결 론

본 논문에서는 다중스레드 모델을 위하여 스레드를 생성할 때 스레드의 길이를 늘이고, 원격 접근 횟수를 감소시키기 위하여 배열지역화 방법을 제안하였다. 본 논문에서 제안한 배열지역화 방법은 프로그램의 루프 액티베이션에서 참조되는 배열의 첨자를 분석하여 해당 루프 액티베이션에서 참조되는 배열의 원소를 루프 액티베이션이 수행되는 노드에 분산 저장한다. 이 방법을 적용하면 스레드의 경계가 되는 원격자료 접근의 수가 줄어들어 스레드의 길이가 증가하고 문맥전환의 숫자가 줄어들기 때문에 시스템의 성능향상을 꾀할 수 있다.

현재 배열이 함수 또는 루프간에 공유될 경우의 지역화에 대해서 연구가 진행되고 있으며, 배열이 이미 처리기 모임의 지역 기억장치에 분산되어 있다고 가정하고 본 논문에서는 실험을 하였으나 앞으로 배열을 각 처리기 모임에 분산 할당하는 실행시간 시스템(run-time system)의 설계가 연구되어야 할 것이다. 그리고 실험 결과의 분석에서 언급한 바와 같이 루프간 종속이 존재하는 경우 접근되는 배열 원소를 지역화할 때 중복 저장되는 배열원소 값의 정의를 위한 원격접근으로 인하여 지역화의 효과가 상쇄되지 않도록 부분 지역화와 완전 지역화사이의 적당한 선택점을 찾는 부분도 추가로 연구되어야 할 것이다.

참 고 문 헌

[1] Arvind and R. A. Iannucci, Two Fundamental Issues in Multiprocessors: The Dataflow Solutions, *MIT/LCS/TR-226-6, Laboratory for Computer Science*, MIT, 1987.

[2] Arvind, R. S. Nikhil, and K. Pingali, "I-Structures : Data Structures for Parallel Computing," *ACM Transactions on Programming Languages and Systems, Vol. 11, No.4*, pp.598-632, 1990.

[3] Arvind and R. E. Thomas, "I-Structures : An Efficient Data Structure for Functional Languages," *MIT/LCS/TM-178*, Laboratory for Computer Science, MIT, Cambridge MA., 1981.

[4] A. P. W. Bohm, "W. A. Najiar, B. Shankar and L. Roh, "An Evaluation of Coarse-grain Dataflow Code Generation Strategies," *Working Conference on Massively Parallel Programming Models*, 1993.

[5] D. E. C. Goldstein, K. E. Schauser, and T. von Eicken, "TAM-A Compiler-Controlled Threaded Abstract Machine," *Journal of Parallel and Distributed Computing, Vol.18, No.3*, pp.347-370, 1993.

[6] J. -L. Gaudiot and L. Bic(editors), *Advanced Topics in Data-flow Computing*, Prentice-Hall, 1991.

[7] V. G. Grafe and J. E. Hoch, "The Epsilon-2 Multiprocessor System," *Journal of Parallel and Distributed Computing, Vol.10, No.4*, pp.309-318, 1990.

[8] J. E. Hochm D. M. Davenport, V. G. Grafe, and K. M. Steele, "Compile-time Partitioning of a Nonstrict Language into Sequential Threads," *Proceedings of 3rd IEEE Symposium on Parallel and Distributed Processing*, pp.180-189, 1991.

[9] R. A. Iannucci, *Parallel Machines : Parallel machine Languages The Emergence of Hybrid Dataflow Computer Architectures*, Kluwer Academic Publishers, 1990.

[10] R. S. Nikhil and Arvind, "Can dataflow subsume von Neumann computing?," *Proc. of 16th Annual Int'l Symp. on Computer Architecture*, pp.262-272, 1989.

[11] R. S. Nikhil, G. M. Papadopoulos, and Arvind, "*T : A Multithreaded Massively Parallel Architecture," *Proc. of 19th Annual Int'l Symp. on Computer Architecture*, pp.156-167, 1992.

[12] L. Rauchwerger and D. Padua, "The Privatizing DOALL Test : A Run-Time Technique for DOALL Loop Identification and Array Privatization," *1994 Int'l Conf. on Supercomputing*, pp.33-43, 1994.

[13] A. Rogers and K. Pingali, "Compiling fo Distributed memory Architectures," *IEEE Transactions on Parallel and Distributed Computing*,

Vol.5, No.3, pp.281-298, 1994.

[14] G. M. Papadopoulos and D. E. Culler, "Monsoon: An Explicit Token-Store Architecture," *Proc. of 17th Annual Int'l Symp. on Computer Architecture*, pp.82-91, 1990.

[15] K. E. Schauser, D. E. Culler, T. von Eicken, "Compiler-Controlled Multithreading for Lenient Parallel Languages," *Proceedings of Symposium on Functional Programming Languages and Computer Architectures, LNCS Vol.523*, pp.50-72, 1991.

[16] S. Sakai, Y. Yamaguchi, K. Hiraki, Y. Kodama, and T. Yuba, "An Architecture of a Dataflow Single Chip Processor," *Proc. of 16th Annual Int'l Symp. on Computer Architecture*, pp.46-53, 1989.

[17] K. R. Traub, "Multithreaded Code Generation for Dataflow Architectures from Nonstrict Programs," *Proceedings of Symposium on Functional Programming Languages and Computer Architectures, LNCS Vol.523*, pp.73-101, 1991.

[18]. K. R. Traub, D. E. Culler, and K. E. Schauser, "Global Analysis for Partitioning Non-strict Programs into Sequential Threads," *Conference on Lisp and Functional Programming*, pp.324-334, 1992.

[19] A. Yoshida, S. Maeda, W. Ogata, and H. Kasahara, "A Data-Localization Scheme for Fortran Macro-Dataflow Computation,"(in Japanese) *Transactions on IECEI, Vol.35, No.9*, pp. 1848-1860, 1994.

양 창 모

1985년 인하대학교 전자계산학과 졸업(이학사)
1988년 인하대학교 대학원 전자계산학과 졸업(이학석사)
1992년~현재 인하대학교 대학원 전자계산학과 박사과정
1990년~현재 동명전문대학 전자계산과 조교수

유 원 희

1975년 서울대학교 공과대학 응용수학과 졸업(이학사)
1978년 서울대학교 대학원 계산학 전공(이학석사)
1985년 서울대학교 대학원 계산학 전공(이학박사)
1979년~현재 인하대학교 공과대학 전자계산공학과 교수
관심분야: 프로그래밍 언어(실시간 프로그래밍 언어, 함수 언어).