KAPPA - KAICUBE II 를 위한 병렬 트랜스레이터

°최 종혁*, 정 봉준*, 조 상영*, 김 탁곤*, 박 규호*
* 한국 과학 기술원 전기 및 전자 공학과

KAPPA - KAICUBE II's Automatic Parallel Programming Aids

Jong-Hyuk Choi*, Bong-Joon Jung*, Sang-Young Cho*, Tag Gon Kim*, Kyu Ho Park*
* Department of Electrical Engineering, Korea Advanced Institute of Science and Technology

KAPPA 는 병렬 프로그램을 보다 쉽고 효율적으로 작성하기 위하여 개발된 병렬 프로그래밍 언어 및 환경이다. 이는 메시지 전달 방식의 병렬처리 컴퓨터인 KAICUBE II 를 위해 개발 되었다. KAPPA는 프로그래머가 문제에 따라 적절히 고안한 병렬 알고리즘에 따라 몇 개의 프로시저로 분할된 순차적 프로그램을 받아 들여 매크로 데이타 흐름 그래프를 만들어 낸다. 이를 바탕으로 스케쥴링, 매핑, 동기화 를 수행하여 병렬 프로그램을 구성 할 수 있는 요소를 추출하여 프로그램 발생기로 하여금 병렬 프로그램을 만들어 낸다. 구해진 병렬 프로그램의 효율성을 분석하기 위하여 시뮬레이터를 사용할 수 있고, 사용자로 하여금 보다 쉽게 효율적인 병렬 프로그램을 작성 할 수 있도록 사용자 인터페이스가 X 윈도우 상에서 동작 하도록 하였다.

I. 서 론

하이퍼큐브 컴퓨터는 메시지 전달에 의한 MIMD 구조로서 많은 수의 프로세서의 연결, 확장이 가능하고 일반적 연결 구조인 선형, 원형, 트리, 메쉬 등으로부터의 매핑 (Mapping) 이 용이하기 때문에 많은 주목을 받아왔다[1]. 한국 과학 기술원 전기 및 전자 공학과 컴퓨터 공학 연구실에서 제작되었던 KAICUBE II 도 이러한 하이퍼큐브 컴퓨터의 한 예이다.

KAICUBE II 에서는 여러 단위 컴퓨터가 같은 프로그램을 수행하면서 각 컴퓨터의 노드 ID 에 따라 자신의 로칼 메모리에 있는 데이타에 대해 프로그램의 다른 부분을 수행하는 형태의 단일 프로그램 다중 데이타 (single program multiple data, SPMD) 형태의 프로그램 기술방식을 택하고 있다[2].

KAPPA (KAICUBE's Automatic Parallel Programming Aids) 는 프로시저 단위의 병렬성을 기초로하여 KAICUBE II 를 위한 병렬 트랜스레이터로서 사용자가 보다 효율적인 병렬 프로그램을 작성할 수 있도록 도와주는 프로그래밍 언어와 환경을 제공한다.

KAPPA 는 프로그래머가 문제에 대해 적절히 고안한 병렬 알고리즘에 따라, 몇 개의 프로시저로 분할된 순차적 프로그램을 받아들인다. 이것은 순차적 프로그램 형태이기 때문에 디버깅을 위해서 순차적 머신에서 수행을 시킬 수도 있다. 이러한 프로그램은 KAPPA 의 전위 부분과 후위 부분을 통해 병렬 코드 합성과 최적화를 통해 메시지 전달 방식인 KAICUBE II 에서 수행될 수 있는 병렬 프로그램으로 바뀐다. 그림 1 은 병렬 트랜스레이터의 전체적인 구성을 보여준다.

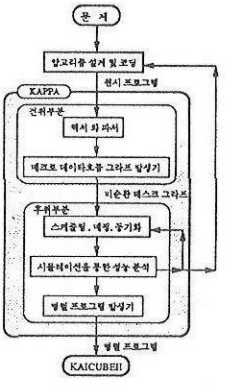

그림 1. 병렬 트랜스레이터의 구성

전위 부분에서는 주어진 프로그램으로부터 프러시저의 인수 (Arguments) 들 간의 데이타 연관성 (Data Dependency) 을 찾아내고 이를 바탕으로 매크로 데이타흐름 그래프 (Macro Dataflow Graph, MDG) 를 만들어 낸다. 후위 부분에서는 전위 부분의 MDG 을 입력으로 받아들여 스케쥴링 (scheduling), 매핑 (mapping), 동기화 (synchronization) 를 수행하여 병렬 프로그램을 구성할 수 있는 요소를 추출한다. 그다음은 이와 같은 결과가 목표 시스템인 KAICUBE II 에서 수행되었을 때 실제적인 수행 시간 및 프로세서들의 사용도를 정확히 알기 위하여 시뮬레이션을 통한 성능 분석을 수행한다. 이 결과를 바탕으로 병렬 프로그램을 재구성할 수 있는 정보를 추출하여 더 좋은 병렬 프로그램을 얻을 수 있다. 성능 분석을 통해 병렬 프로그램의 구성이 확정되면 최종적으로 병렬 프로그램 발생기 (Code Generator) 가 병렬 프로그램을 만들

어낸다.

II. 전위 부분

2.1. 언어 설계

KAPPA 프로그래밍 언어는 C 프로그래밍 언어를 기초로 하여 설계되었다. KAPPA 는 ANSI C 의 모든 문법을 지원하며, 여기에 프로그램의 병렬성 (Parallelism) 을 나타내기 위해 필요한 요소들을 첨가하였고, 병렬 프로그래밍을 위한 프로그래밍 스타일을 제공하였다.

그림 2 에 KAPPA 의 원시 프로그램 (Source Program) 을 예로서 나타내었다. 이는 가우스 소거법 알고리즘 (Gaussian Elimination Algorithm) 을 병렬화 한 프로그램으로, 주어진 행렬을 행 (Column) 단위로 분할한 것이다 [4].

앞의 예제 프로그램에서 알 수 있듯이 구문적 요소 (Lexical Construct) 로서 "par" 라는 예약어 (Keyword) 를 첨가하였다. 이 예약어는 함수 선언 (Function Definition) 앞에 붙어서, 주어진 함수가 병렬 함수 (Parallel Function) 임을 나타내고 있다. 여기에서 병렬 함수란 더 이상 나누어지지 않는 스케쥴링의 단위를 나타내며, 이는 사용자에 의해 분할된 하나의 프러시저이다.

KAPPA 의 원시 프로그램은, main 함수에서 call 되는 병렬 함수들로 구성되어 진다. 이같은 프로그래밍 스타일은 그림 2 에서 알 수 있듯이 Modularity 가 좋고, 통신 프리미티브가 쓰이지 않기 때문에 목적 하드웨어 (Target Architecture) 에 독립적인 프로그래밍을 할 수 있다는 장점이 있다.

```
#define N 4

typedef struct vector_tag {
        int     index[N];
        float   w[N];
} Vector;

float   matrix[N+1][N+1][N];
Vector  vector[N+1];
float   Answer[N];

void InitMtx()                                  // Serial Function
{
}

void BackSubs()                                 // Serial Function
{
}

void PrintMtx()                                 // Serial Function
{
}

par FindMax(inColumn, inVec, outVec, k)         // Parallel Function
float   inColumn[];
Vector  inVec, *outVec;
int     k;
{
}

par UpdateMtx(inColumn, outColumn, inVec, k)    // Parallel Function
float   inColumn[], *outColumn;
Vector  inVec;
int     k;
{
}

main()
{
        int     i,j;

        InitMtx();
        for ( i = 0; i < N; i++ ) {
                FindMax(matrix[i][i], vector[i], &vector[i+1], i);
                for ( j = i; j < 5; j++ )
                        UpdateMtx(matrix[i][j], matrix[i+1][j], vector[i+1],i);
        }
        BackSubs();
        PrintMtx();
```

그림 2. KAPPA 의 입력 프로그램

2.2. 매크로 데이타 흐름 그래프 (Macro Dataflow Graph, MDG)

그림 3 에 앞의 예제 프로그램에 대한 MDG 를 나타내었다. 이 MDG 의 각 노드 는 각 병렬 함수의 수행에 해당하고, 이 노드 의 값은 병렬 함수의 해당 수행의 수행 시간 을 나타 낸다. 각 에지는 병렬 함수간에 전송되는 메시지에 해당되며, 에지의 값은 메시지의 전송시간을 나타낸다.

MDG 를 생성하기 위하여, 먼저 제어 흐름 그래프 (Control Flow Graph, CFG) 를 만들고, 이 CFG 에 따라서 MDG 를 생성한다. 이 때에, Dynamic 한 Code에 대하여는 compile-time 에 CFG 를 만들기가 어려우므로, KAPPA 는 Static 한 Code 에 대하여만 CFG 를 만들고, 이에 따라 MDG 를 생성한다.

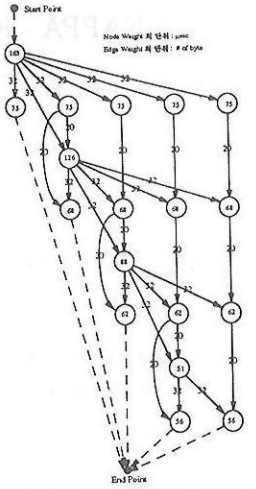

그림 3. 예제 프로그램에 대한 매크로 데이타 흐름 그래프

각 노드의 수행 시간을 구하기 위해 KAICUBE II 상에서 UNIX 의 Prof Utility [5] 를 사용하여, 기본적인 연산과 Reference Pattern 에 대해 수행 시간을 데이타베이스로 만들고, 이에 기초하여 원시 프로그램에 대해 기본적인 연산과 각 Reference Pattern 에 대해 수행 시간을 구한다. 수행 시간의 측정 역시 Static 한 Code 에 대해서만 의미가 있다.

각 에지의 값, 즉 통신 시간 (T_C) 은 다음 식에 의하여 계산된다.

$$T_C = T_S + \frac{M_{Size}}{B}$$

위의 식에서 T_S 는 통신 시작 시간으로 메시지를 보내고 받기 위해 준비하는데 걸리는 시간이고, M_{size} 는 전송되는 메시지의 양 (Size), B 는 Bandwidth 이다.

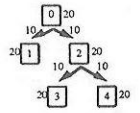

그림 4. Macro Dataflow Graph

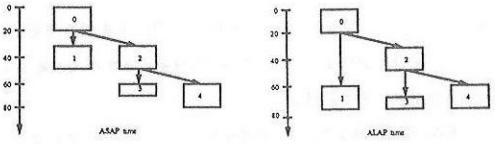

그림 5. 그림 4 의 MDG 에 대한 ASAP & ALAP 시간

III. 스케쥴링

원시 프로그램이 KAPPA 의 전위부분을 거쳐 MDG 가 생성되면, 이것의 각 노드를 특정한 PE (Processing Element) 에 적절히 배분을 해서 전체 수행시간이 최소가 되도록 수행순서를 결정해야 한다. KAPPA 에서는 최적화된 결과를 얻기 위해서 MD (Mobility Directed) Scheduling Algorithm 을 사용하였다.[3][6]

아래에 MD Scheduling Algorithm 과 여기에서 사용된 용어와 조건을 기술하였다.

ASAP 시간은 각 노드들을 가능하면 빨리 수행되도록 하였을때의 각 노드의 수행 시작 시간이며, ALAP 시간은 각 노드들을 가능하면 늦게 수행되도록 하였을때의 각 노드의 수행 시작 시간이다. (그림 5) Mobility 란 각 노드들의 ALAP 시간에서 ASAP 시간을 뺀 값, 즉 움직일 수 있는 범위를 나타낸며, Relative Mobility 는 각 노드들의 Mobility 에서 그 노드 의 값을 나눈 값으로서 스케쥴링할 순서를 정하는 기준이 된다.

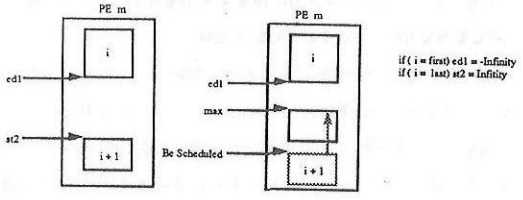

그림 6. (a) Scheduling Condition
(b) Advance Scheduling Time

- 스케쥴링 조건

1) 같은 PE 에서 스케쥴링 될 수 있는 조건 (그림 6a).

Pre-condition

1. st2 - ed1 > Node weight(n)
2. st-time ≤ ALAP(n), where st-time = max(ed1, ASAP(n))
3. st-time ≤ st2

Post-condition

1. st-time + Node weight(n) ≤ st2
2. schedule 하게되면 back edge 가 생길경우

2) 가장 빨리 스케쥴링 될 수 있는 시간 계산 (그림 6b).

max = ed1

foreach Node which has edge to this node

ttime = Node 의 시작시간 + edge weight + node weight(n)

if(ttime > max) max = ttime

- MD (Mobility Directed) Algorithm

while(there exist not selected node)

Calculate relative mobility for all node

Ni = Minimum Relative mobility and Not selected

스케쥴링 조건 1) 에 의해 PE 0 에서부터 차례대로 스케쥴링 할 수 있는지 조사.

PE m 에 노드가 스케쥴링 되었다면 스케쥴링 조건 2) 에 의해 가장 빨리 스케쥴링 할 수 있는 시간 계산, 그 시간에 스케쥴.

Mark Ni to Selected

IV. 매핑

스케쥴링에서 만들어진 PE 가 노드가 되고 각 PE 사이의 통신량을 모두 합한 것이 PE 사이의 에지가 된 그래프를 만들어 이를 태스크 그래프 (Task Graph) 라 한다. 이 태스크 그래프를 시스템 그래프 (System Graph: 실제 하드웨어의 연결 상태를 나타내는 그래프) 에 어떻게 대응시키면 전체의 통신량이 줄어들 것인가를 정하는 것이 매핑이다. 예로서 그림 7 과 같은 태스크 그래프와 시스템 그래프가 있다면 첫째와 같이 매핑 했을 때의 총 통신량은 3 + 2 + 3 + 6 × 2 = 20 이었으나, 두번째와 같이 매핑 하게되면 총 통신량은 3 + 6 + 2 × 2 = 16 으로 줄어듦을 알 수 있다. 실제로 구현하는 알고리즘은 Kernighan 과 Lee 에 의해서 제안한 방법에 기초를 두고 하이퍼큐브 컴퓨터 구조에 맞게 분할하는 방법을 고안하여 만들었다.[7]

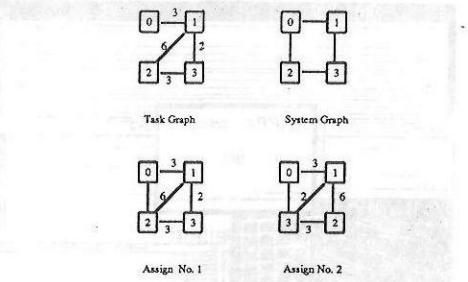

그림 7. 매핑

V. 동기화

매핑이 끝난 다음의 각 노드와의 통신은 Send 나 Receive 의 통신 프리미티브에 의해 이루어 진다. 만약 PE 0 에서 PE 1 으로 에지가 있으면 PE 0 에서는 Send 프리미티브가 수행되고 PE 1 에서는 Receive 프리미티브가 수행되어야 한다.

KAICUBE II 에서는 OS 차원에서 Typed-Message 방식을 제공하여 주므로 통신순서를 일치시키지 않고서도 위의 문제를 해결할 수 있다. Typed-Message 방식이란 각각의 메시지에 고유번호를 부여하고, Send 프리미티브가 특정한 번호로서 보내면 받는 쪽에서는 Receive 큐에 저장해 두었다가 Receive 프리미티브가 실행될때 마다 같은 특정한 Type 만을 골라 받을 수 있도록 하는 방식이다.

VI. 시뮬레이터

보통 스케쥴링 알고리즘은 하드웨어의 많은 부분을 간단하게 가정하고 수행하기 때문에 알고리즘적으로 최적의 해를 구했다 하더라도 이는 실제의 수행 상태를 정확히 예견할 수 있는 최적해는 아니다. 이를 실제 시뮬레이션을 통해 결과의 수행을 분석하면 실제적으로 더 좋은 스케쥴링을 수행할 수 있다. 즉, 알고리즘적으로 구한 결과를 초기 결과로 하고 시뮬레이션을 통해 계속 보완해 갈 수 있는 방식을 만들었다. 더욱이 시뮬레이터를 통한 결과는 문제에 대한 근본적 알고리즘을 개선하는데에도 사용될 수 있다.

시뮬레이터가 제공하는 정보는 전체적 속도 증가와 효율성, 각 프로세서의 사용도, 통신 프리미티브의 수행과정 등이다. 만약 어떤 통신 프리미티브가 잘못 넣어져 시뮬레이션 결과 전체 수행속도에 지장을 주고 있다면 이것을 적절한 위치에 넣을 수 있으며 필요하다면 각 모듈의 위치도 정하여 줄 수 있다.

VII. 사용자 인터페이스

KAPPA 의 사용자 인터페이스는 X 윈도우 상에서 동작하도록 작성되었다. 주 윈도우 상에서 여러 부 윈도우들을 띄워 원시 프로그램을 작성, 수정하고, KAPPA 의 전반부를 수행시켜 MDG 를 생성하고, 생성된 MDG 를 사용

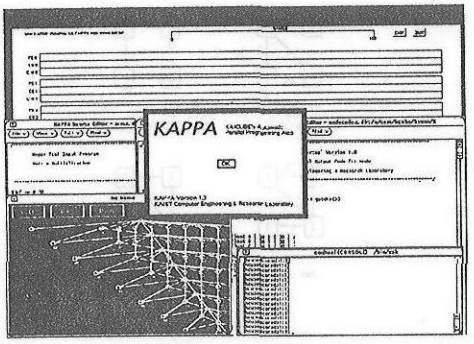

그림 8. KAPPA 의 사용자 인터페이스

자에게 보여주고, 스케쥴링, 매핑, 그리고 동기화를 수행하여 그 결과를 사용자에게 보여주고, 시뮬레이션을 수행하여 그 결과를 보여주는 등의 모든 인터페이스를 제공하였다 (그림 8). 이렇게 편리하고 사용자에게 많은 정보를 제공하는 인터페이스를 통해 사용자가 보다 쉽게 효율적인 병렬 프로그램을 작성할 수 있도록 하였다.

VIII. 목적 프로그램 생성

KAPPA 프로그래밍 언어에서의 목적 언어는 하이퍼큐브 멀티컴퓨터인 KAICUBE II 에서 수행되는 SPMD 형태의 병렬 프로그램이다. KAPPA 는 호스트를 위한 프로그램과 노드를 위한 프로그램을 각각 생성한다.

```
#include <hcube/node.h>
#define ____NODES       (1 << getdim())
#define ____T_START     1
#define ____T_DOWN      2
#define ____T_ACK       3
#define ____MYUID       1

typedef struct vector_tag {
        int index [ 4 ] ;
        float m [ 4 ] ;
} Vector ;
float matrix [ 4 * 1 ] [ 4 * 1 ] [ 4 ] ;
Vector vector [ 4 * 1 ] ;
float Answer [ 4 ] ;

FindMax ( inColumn , inVec , outVec , k )
float inColumn [ ] ; Vector inVec ; Vector * outVec ; int k ;
{
}

UpdateMtx ( inColumn , outColumn , inVec , k )
float inColumn [ ] ; float * outColumn ; Vector inVec ; int k ;
{
}

main()
{
        int     ____snode,____supid,____hostpid,____nid,____i,____data;

        if (____nid == 0) {
                recvw(&____snode,&____supid,____T_DOWN,matrix[0][0],sizeof(matrix[0][0]));
                recvw(&____snode,&____supid,____T_DOWN,&vector[0],sizeof(vector[0]));
                recvw(&____snode,&____supid,____T_DOWN,matrix[0][1],sizeof(matrix[0][1]));
                FindMax(matrix[0][0],vector[0],&vector[1],0);
                sendw(1,____MYUID,23,&vector[1],sizeof(vector[1]));
                sendw(2,____MYUID,22,&vector[1],sizeof(vector[1]));
                sendw(1,____MYUID,21,&vector[1],sizeof(vector[1]));
                sendw(1,____MYUID,20,&vector[1],sizeof(vector[1]));
                UpdateMtx(matrix[0][1],matrix[1][1],vector[1],0);
                sendw(1,____MYUID,24,matrix[1][1],sizeof(matrix[1][1]));
                FindMax(matrix[1][1],vector[1],&vector[2],1);
                sendw(2,____MYUID,30,&vector[2],sizeof(vector[2]));
                sendw(1,____MYUID,29,&vector[2],sizeof(vector[2]));
                sendw(1,____MYUID,28,&vector[2],sizeof(vector[2]));
                recvw(&____snode,&____supid,25,matrix[1][2],sizeof(matrix[1][2]));
                UpdateMtx(matrix[1][2],matrix[2][2],vector[2],1);
                sendw(1,____MYUID,31,matrix[2][2],sizeof(matrix[2][2]));
                FindMax(matrix[2][2],vector[2],&vector[3],2);
                sendw(1,____MYUID,34,&vector[3],sizeof(vector[3]));
                recvw(&____snode,&____supid,32,matrix[2][3],sizeof(matrix[2][3]));
                UpdateMtx(matrix[2][3],matrix[3][3],vector[3],2);
                FindMax(matrix[3][3],vector[3],&vector[4],3);
                recvw(&____snode,&____supid,33,matrix[2][4],sizeof(matrix[2][4]));
                UpdateMtx(matrix[2][4],matrix[3][4],vector[3],2);
                UpdateMtx(matrix[3][3],matrix[4][3],vector[4],3);
                UpdateMtx(matrix[3][4],matrix[4][4],vector[4],3);
                sendw(HOSTPID,____hostpid,10,matrix[4][3],sizeof(matrix[4][3]));
                sendw(HOSTPID,____hostpid,11,matrix[4][4],sizeof(matrix[4][4]));
                sendw(HOSTPID,____hostpid,15,&vector[4],sizeof(vector[4]));
        }
        else if (____nid == 1) {
        }
        else if (____nid == 2) {
        }
}
```

그림 9. Gaussian Elimination 프로그램의 Node Target Code

여기서는 노드프로그램만 그림 9 에 예시하였다. 노드 목적 프로그램은 우선 동기를 위하여 T_START 타입의 메시지를 호스트로 부터 받고, T_ACK 타입의 메시지를 호스트로 보낸다. 동기가 다 된 후에, 호스트로부터 필요한 데이타를 다운로드받고, 병렬 함수의 실행을 시작하며, 필요한 데이타를 다른 노드들과 주고 받는다. 실행이 다 끝난 후 계산의 결과를 호스트에게로 업로드 한다.

이렇게 생성된 호스트 프로그램과 노드 프로그램은 KAICUBE II 상의 범용 C 컴파일러로 컴파일 되어서 KAICUBE II 상에서 수행된다.

IX. 결론 및 연구 과제

KAPPA 는 병렬 프로그램을 보다 쉽고 효율적으로 작성하기 위해 개발된 병렬 프로그래밍 언어 및 프로그래밍 환경이다. KAPPA 는 메시지전달 방식의 병렬처리 컴퓨터인 KAICUBE II 를 위해 개발되었으며, 어떤 형태의 메시지전달형 병렬처리 컴퓨터에도 이식할 수 있다.

KAPPA 에서 제공하는 것과 같은 프로그래밍 스타일은 매우 일반적인 방법이다. 예를 들어, MUPPET, POKER, Polylith 그리고 Hypertool 과 같은 병렬 프로그래밍 환경에서도 KAPPA 의 프로그래밍 스타일과 같은 프로그래밍 스타일을 이용하고 있다.[6] KAPPA 의 사용 결과 프로그래밍 스타일이 만족할 만큼 일반적이었으며, 생성된 프로그램의 효율도 선택된 어플리케이션과 알고리즘에 따라 차이는 있지만, 대체적으로 매우 높았다.

앞으로의 연구로는 메시지전달형 병렬처리 컴퓨터에서의 프로그래밍을 위한 보다 나은 프로그래밍 스타일의 개발이 필요하다.

참고 문헌

[1] Y. Saad and M.H. Schultz, "Topological properties of hypercube," IEEE Trans. Comput., pp.458-473, May 1977.

[2] A.H. Karp , "Programming for parallelism," IEEE Comput. vol. 20, pp.43-57, May 1987.

[3] M.-Y. Wu and D.D. Gajski, "Hypertool: a programming aid for message-passing systems," IEEE Trans. on Parallel and Distributed Systems vol. 1, No. 3, pp.330-343, 1990.

[4] Y. Robert, "The Impact of Vector and Parallel Architectures on the Gaussian Elimination Algorithm," Manchester University Press, ch.1-4, 1990.

[5] AT & T, " UNIX SYSTEM V RELEASE 4 Programmer's Guide : ANSI C and Programming Support Tools," Prentice Hall, ch.7, 1990.

[6] A. W. Kwan, L. Bic, and D. D. Gajski, "Improving parallel program performance using critical path analysis," Languages and Compilers for Parallel Computing, D. Gelernter, A. Nicolau, and D. Padua Editors, pp.358-373, Pitman, 1990.

[7] C.-H. Lee, C.-I. Park, and M. Kim, "Efficient algorithm for graph partitioning problem using a problem transformation method," Computer-Aided Design, vol. 21, pp.611-618, Dec. 1989.