韓國經營科學會誌
第18卷 第3號
1993年 12月

Lagrangean 근사과정의 병렬계산†

이호창*

On Parallel Implementation of Lagrangean Approximation Procedure

Hochang Lee*

Abstract

By operating on many parts of a software system concurrently, the parallel processing computers may provide several orders of magnitude more computing power than traditional serial computers. If the Lagrangean approximation procedure is applied to a large scale manufacturing problem which is decomposable into many subproblems, the procedure is a perfect candidate for parellel processing. By distributing Lagrangean subproblems for given multiplier to multiple processors, concurrently running processors and modifying Lagrangean multipliers at the end of each iteration of a subgradient method, a parallel processing of a Lagrangean approximation procedure may provide a significant speedup.

The purpose of this research is to investigate the potential of the parallelized Lagrangean approximation procedure(PLAP) for certain combinatorial optimization problems in manufacturing systems. The framework of a PLAP is proposed for some combinatorial manufacturing problems which are decomposable into well-structured subproblems. The synchronous PLAP for the multistage dynamic lot-sizing problem is implemented on a parallel computer Alliant FX/4 and its computational experience is reported as a promising application of vector-concurrent computing.

† 이 연구는 1992년도 한국과학재단 연구비지원에 의한 결과임(과제번호 : 923-0900-012-1)
* 경희대학교 사회과학대학 경영학과

1. 서 론

생산시스템에서 흔히 보게되는 현실적인 조합적(combinatorial) 최적화문제들은 그 크기가 매우 클뿐만아니라 계산적으로도 복잡하여 실시간(real time)에 해를 제공해야 하는 모델의 실용성 관점에서 볼때 종래의 직렬계산 처리방식을 이용할 경우 그 활용도면에서 많은 제약을 받아왔다. 그러나 이 최적화문제들을 자세히 살펴보면 제약조건들의 일부분을 완화(relax)하면 대부분의 경우 기존의 효율적인 알고리즘에 의하여 쉽고 빠르게 해들을 얻을수 있는 소규모의 부문제(subproblem)들로 분해가 가능하다는사실을 발견하게 된다. 전형적인 예들로는 다품목 롯사이징문제, 다품목 공장입지문제, 다단계 분배문제 등과 같은 다차원(multiechelon)의 생산문제를 들수있는데 이들의 경우 단일품목, 개별단계의 부문제들이 품목간 또는 단계간의 동시셋업(joint setup)이나 물류보전방정식(flow balance equation)등과 같은 제약조건들로 연결되어진다. 이러한 문제들의 전형적인 해법은 크게 다음의 세가지 절차들로 구성된다. 첫째, 연결제약조건들을 완화함으로써 거대한 원문제를 소규모의 부문제들로 분해한다. 둘째, 부문제들의 특수한 구조를 이용하여 그 해들을 구하거나 근사한다. 세째, 부문제들로부터 얻은 부분해들을 완화된 제약조건을 만족시키도록 통합, 조정하는 주문제(master problem)를 푼다. 주문제의 형태에는 예를 들어 cross decomposition[25]과 같은 반복적인(iterative) 알고리즘에서보는 정량적(quantitative) 최적화문제뿐만 아니라 모델링할 수 없는 최종 결정자들의 정성적(qualitative)인 의사조정절차도 포함시킬 수 있다. 정량적 주문제의 경우 생산조직의 최종목적, 해의 근사정도, 각부문간의 조화등과 같은 생산환경 및 관리자의 필요성에 따라서 그해법의 방향을 달리한다. 본 연구에서 다루는 Lagrangean 근사과정은 부문제의 근들을 통합 조정하는 정량적 주문제의 한 형태이다. 총괄생산계획문제의 대별되는 두 해법[14]들은 서로 다른 주문제 형태들을 보여주기에 충분하다. 첫째로 거대한 단일해법(monolithic approach)은 총괄 생산계획문제를 하나의 정수계획모형으로 정식화하고 이에 Lagrangean 완화기법을 적용하여 분해된 단일기간의 문제들을 풀어 원문제의 해를 근사한다. 이에 반해 계층해법(hierarchical approach)은 원문제를 생산계획의 시간적 선후관계나 결정문제의 중요성에 따라 계층적 부문제들로 정식화하고 순차적으로 풀어서 상위문제의 해가 하위문제에 대한 새로운 제약조건이 되도록한다. 위의 두해법에서 보는바와 같이 부문제들이 각 단계나 계층에서 분리되어 독립적으로 풀리는 방식은 동일하나 주문제에서 부문제들의 해가 통합, 조정되는 방식은 서로 다르다. 단일해법의 주문제에서는 subgradient method를 이용한 반복적 계산단계에서 Lagrangean dual 문제를 최적화하기위한 Lagrangean 승수들이 수정되며 계층해법의 주문제에서는 각 계획 부문제들간의 상호작용 및 각계층의 의사결정자들에 의하여 조직전체의 목적을 조화, 개선해가는 방향으로 부문제들의 해가 조정된다.

Lagrangean 근사과정은 원문제의 제약조건 일부를 완화하여 얻은 Lagrangean 완화문제를 푸는 일종의 한정기법(bounding technique)이다. 여기에 Lagrangean 휴리스틱을 가미하면 다른 한쪽의 상한이나 하한을 제공하여 원문제의 최적해에 대한 근사해법이 된다. 미분 불가

능한 Lagrangean dual 함수를 최적화하기 위해서는 몇가지 종결조건들을 만족시킬때 까지 Lagrangean 승수들을 조정하는 subgradient method를 이용한 반복계산단계가 DO-loop의 형태로 진행되며 각단계마다 분해된 부문제들이 주어진 승수에 대하여 독립적으로 최적화된다.

병렬처리컴퓨터는 수개에서부터 수천개에 이르는 독립적인 프로세서들로 구성되며 이들은 상호 데이터의 교환 및 통화를 위하여 각 기종마다 고유한 형태의 망구조상에서 메모리들과 연결된다. 병렬컴퓨터의 구조를 결정짓는 요소들중의 하나는 프로세서들간의 정보교환방식인데 이에는 크게 공유메모리(shared memory)와 분산메모리 (distributed memory)의 두가지 극단적인 형태들이있다. 전자에서는 모든 프로세서들이 하나의 메모리를 공유함으로써 이에 직접 접근할 수 있기때문에 이 공유메모리에 기록하고 읽음으로써 프로세서들간의 통화가 가능하다. 후자에서는 각 프로세서들이 자신만을 위한 전용메모리를 가지고 있으므로 이들을 연결하는 통신망을 통하여서만 정보교환이 가능하다. 이들 양극단적인 구조들의 절충형태가 존재함은 물론이며 이들에 대한 자세한 구조분석은 Ribeiro[23]와 Bertsekas 와 Tsitsiklis[3]를 참조하라.

소프트웨어 시스템의 병렬화는 크게 두가지 방식으로 진행되는데[28] 하나는 알고리즘병렬화(algorithm parallelism)이고 다른 하나는소프트웨어병렬화(software parallelism)이다. 전자는 알고리즘 자체를 병렬개념으로 새로 고안하는 방식으로 대응하는 기존의 직렬알고리즘과는 처리순서 및 내용을 전혀 달리하는데 반해 후자는 기존의 직렬알고리즘을 사용하고자 하는 병렬컴퓨터의 특수한 구조에 맞게 재구성하고 다듬는 과정(streamlining)을 말한다. 소프트웨어병렬화는 사칙연산과 같이 매우 기본적인 작은 계산량에서부터 독립적인 부문제들의 최적화과정과 같은 대규모 계산량에 이르기까지 각 프로세서에 할당되는 수행작업량의 크기 (granularity of task allocation)에 따라 구분될 수 있는데 병렬처리효율을 최적화하기 위해서는 알고리즘의 고안이 분해된 부작업들의 크기(degree of granularity) - 대분(coarse granulated), 혹은 세분(fine granulated) - 사용하려고 하는 병렬컴퓨터의 구조(프로세서의 갯수 및 각 프로세서의 연산능력, 통신망의 형태등)에 따라 결정되어야 함은 당연하다. 최근 대규모 생산스케쥴링문제를 위한 병렬계산의 응용[4]은 대분된 소프트웨어 병렬화의 한 예이며 본 연구에서 다루는 Lagrangean 근사과정의 병렬화도 이 범주에 든다.

병렬처리컴퓨터는 하나의 소프트웨어시스템을 여러부분으로 쪼개어 다수의 프로세서에서 분산처리함으로써 기존의 직렬처리 컴퓨터와는 비교가 되지않는 계산능력을 발휘한다. 약간의 제약조건들을 완화할 경우 많은 부문제들로 분해 가능한 대규모 생산문제에 응용된 Lagrangean 근사과정은 병렬처리에 매우 적합한 계산구조를 갖는다. 주어진 Lagrangean 승수에 대하여 분해된 부문제의 최적화문제들을 각 프로세서들에 할당하여 동시계산하고 subgradient method의 매 단계마다 부문제들의 최적해를 취합하여 다음단계의 승수를 결정하는 주문제로 구성되어지는 병렬알고리즘은 직렬알고리즘과 비교할때 상당한 계산속도상의 가속효과(speedup)를 나타낸다. 본 연구의 목적은 생산시스템의 대규모 조합적(combinatorial) 최적화문제를 위한 병렬 Lagrangean 근사과정을 제시하고 계산처리속도면에 있어서

그 가능성을 타진하는 것이다.

2. 생산 시스템의 조합적 최적화문제의 분해

본 장에서는 Lagrangean 근사과정의 응용대상인 생산시스템의 주요 대규모 조합적(combinatorial) 최적화문제들이 어떻게 특수한 해법을 갖는 소규모 부문제들로 분해되어지는지를 살펴봄으로써 이들 문제에 대한 병렬계산의 적합성을 강조한다. 분해과정의 표현상 공통적으로 사용되어질 수학적 기호를 정의하면 주어진 최적화문제 (P)에 대해서 OV(P)와 OS(P)는 각각 (P)의 최적치와 최적해를 말하며 u,v,w는 완화된 제약조건에 대응하는 Lagrangean 승수백터들을 나타낸다. 또한 (P) → $\Sigma_i(P_i)$ 는 원문제 (P)가 부문제 (P_i)들로 분해됨을 나타낸다. 각 정식과 이에 쓰여지는 변수들의 정의는 인용된 문헌의 사용예와 동일하므로 이들의 자세한 설명은 생략하고 분해 결과만을 간략히 설명한다.

2.1 총괄생산계획문제 (Aggregate Production Planning Problem)

일괄생산방식에 있어서 총괄생산계획이라함은 최소한의 생산비용으로 소비자의 요구를 충족시켜주기위한 생산자원의 확보, 이용 및 분배계획을 수립하는 것을 말한다. 전형적인 의사결정사항들로는 인력수준, 잔업계획, 벳치사이즈결정 및 그들의 공정순서등을 들수있다. Graves[14]는 Hax와 Meal이 제시한 생산구조개념(production framework)하에서 부문별 생산부문제들과 미시적인 공정순서 결정문제들을 하나의 거대한 혼합정수계획모형으로 정식화하고 이의 최적화기법으로 단일해법과 계층해법을 교차하여 사용하는 혼합해법을 제시하였다. 제품종류 i와 제품군 j를 연결시켜주는 제한조건을 완화하여 얻게되는 Lagrangean 완화문제 (LR1(v))는 주어진 Lagrangean 승수 v에 대해서 아래와 같은 소규모 부문제들로 분해된다.

$$(LR1(v)) \rightarrow (LR11(v)) + (LR12(v)) \rightarrow (LR11(v)) + \sum_j (LR12_j(v))$$

여기서 (LR11(v))는 제품종류에 대한 계획모형이고 (LR12(v))는 제품군에 대한 계획모형인데 특히 (LR11(v))는 선형계획모형이고 각 제품군 j에 대한 ($LR12_j(v)$)는 동적계획법과 같은 polynomial 알고리즘에 의하여 손쉽게 풀려지는 무한용량롯사이징(uncapacitated lot-sizing)문제이다.

2.2 다품목 단순공장입지문제 (Multiproduct Simple Plant Location Problem)

다품목 단순공장입지문제(MSPLP)는 다품목의 수요를 만족시키는 제한조건들이 첨가된 단순공장입지문제(SPLP)의 확장형문제이다. MSPLP는 공장을 여는데 드는 고정비이외에도 특정한 제품을 생산하는데 드는 품목별고정비 뿐만아니라 소비자, 품목 및공장에 따라 발생하는 소비자-품목-공장별 수송비용을 목적함수인 총생산비에 포함한다. 특히 MSPLP는 품목별 고정비가 상대적으로 높은 생산시스템의 공장입지 선정문제에 유용하게 활용된다. Klince-

wicz et al. [18]가 정식화한 MSPLP의 Lagrangean 완화문제 는 주어진 승수 v에 대하여 다음과 같이 분해된다.

$$(LR2(v)) \rightarrow (LR21(v)) + (LR22(v)) \rightarrow (LR21(v)) + \sum (LR22(v))$$

여기서 (LR21(v))는 목적함수 계수의 부호에 따라서 해가 자동적으로 결정되는 손쉬운 최적화문제이며 각 생산품목 i에 대한 부문제 (LR22(v))는 DUALOC 알고리즘[7]으로 쉽게 풀수있는 단순공장입지문제이다.

2.3 한정용량 집괴문제(Capacitated Clustering Problem)

한정용량 집괴문제(CCP)는 각 집단마다의 크기제한조건하에서 자료 혹은 개체들간의 동질성(homogeneity)및 개체들로 구성된 집단간의 이질성(heterogeneity)을 최대화하도록 자료들을 덩어리로 묶는(grouping) 문제이다. 통신망설계의 근간을 이루는 한정용량 집중기위치선정문제(capacitated concentrator location problem : CCLP)[22]도 이 한정용량 집괴문제의 변형으로 볼수있는데 이는 집중기의 처리용량하에서 통신망상 그 위치들을 정하고 지역적으로 분산되어 위치한 통신수요점들을 개개의 집중기에 연결하는 통신비용 최소화문제이다. Mulvey와 Beck[21]이 정식화한 CCP의 Lagrangean 완화문제 (LR3(v))는 각 집괴 j에 대하여 FPK79[8]와 같은 pseudopolynomial 알고리즘에 의하여 효과적으로 풀리는 배낭문제 (LR3(v))로 분해된다.

$$(LR3(v)) \rightarrow \sum (LR3(v))$$

2.4 다단계 동적롯사이즈문제 (Multistage Dynamic Lot-sizingProblem)

다단계 조립공정시스템에서는 여러종류의 반제품들이 다른 하나의 중간제품으로 조립되고 이는 또다시 다음단계의 반제품 조립과정에서 요구되는 여러가지 부분품중의 하나로 쓰여지게된다. 다단계 동적롯사이즈문제(MDLP)는 이러한 다단계 조립공정시스템하에서 최소의 비용으로 최종생산품에 대한 소비자의 수요를 만족시키는데 필요한 모든 중간단계의 반제품 및 원료의 생산량과 재고량들을 결정하는것이다. 계산의 관점에서 볼때 MDLP에 대한 종래의 혼합정수 최적화모형은 그 본래의 조합적 구조(combinatorial structure)때문에 극히 난해한 문제이지만 Afentakis et al. [1]는 쉽게 분해되는 단계별 재고(echelon stock)를 이용하여 재정식화하였다. 그들의 MDLP에 대한 Lagrangean 완화문제 (LR4(v))는 각 반제품 i마다 단일품목 롯사이징문제 (LR4(v))들로 분해된다.

$$(LR4(v)) \rightarrow \sum (LR4_i(v))$$

2.5 동시자원계획문제 (Simultaneous Resource Scheduling Problem)

일괄생산시스템에서 여러공정(job)들이 동시에 한가지 이상의 생산도구나 기술과 같은 생산자원들을 필요로하는 경우가 발생하는데 이러한 생산자원의 양이 제한되어있는 경우 각 공정들에 대해서 생산자원의 분배 및 이용계획

의 수립이 필요하다. Dobson과 Karmarkar[6]는 다수의 공정과 생산자원하에서 가중흐름시간(weighted flow time)을 최소로하는 동시자원계획문제(SRSP)를 정식화하였는데 이의 Lagrangean완화문제 (LR5(v))는 아래와 같이 공정 j에 대하여 단일기계의 가중흐름시간 최소화문제 ($LR5_j(v)$)로 분해된다.

$$(LR5(v)) \rightarrow \sum_j (LR5_j(v))$$

2.6 차량경로문제 (Vehicle Routing Problem)

차량경로문제(VRP)는 중앙차고(central depot)에 위치한 차량들이 지역적으로 분산되어 있는 고객들의 수요를 충족시키기 위하여 운행될때 최소한의 차량운용비용이 드는 경로를 결정하는 것이다. 이는 학교통학버스나 각지에 소비재를 공급하는 수송분배문제등에서 그 예를 흔히 볼수있다. Fisher와 Jaikumar[9]이 정식화한 비균일비대칭 차량경로문제(non-uniform fleet asymmetric VRP)의 Lagrangean 완화문제 (LR6(u,v,w))는 승수 u,v,w에 대하여 다음과 같이 분해된다.

$$(LR6(u,v,w)) \rightarrow \sum_k (LR6_k(u,v,w))$$

여기서 각 차량 k에 대하여 ($LR6_k(u,v,w)$)는 0-1 배낭문제이다.

3. 병렬컴퓨터와 병렬알고리즘

병렬분산(parallel distributed)처리에는 상대적 개념인 직렬(serial)처리와 대비할때 몇가지 부가적으로 고려해야할 사항들이 있는데 그중 하나가 작업배분(task allocation)의 문제이다. 이는 어떻게 전체작업을 소규모의 부작업들로 구분하여 각 프로세서에 처리를 할당하며 일부 부작업들이 동시에 서로 다른 프로세서에 의하여 독립적으로 행하여질 수 없는 계산종속의 경우 어떻게 효율적인 계산처리 계획을 수립하는가 하는 것이다. 두번째로 분산된 프로세서들간의 중간결과들에 대한 통화(communication)와 그들에 의하여 수행되는 계산의 동시화 및 비동시화(synchronization and asynchronization) 문제이다. 비동시계산은 어떤 일정한 시점에서 특정한 프로세서가 필요한 자료의 도착이나 다른프로세서의 처리종료를 기다릴 필요가 없는 독립적인 계산방식을 말한다. 병렬처리의 수행효율을 측정하는 모델과 특정한 병렬처리시스템의 구조가 수행효율에 미치는 영향분석도 새로이 대두되는 연구과제들중의 하나이다.

3.1 병렬시스템의 구조와 프로세서들간의 통화

일반적으로 병렬처리시스템의 구조를 분류하는데 몇가지 기준들이 있는데 그중 하나가 시스템을 구성하는 프로세서의 형태와 그 갯수이다. 세분된(fine grained) 병렬시스템은 수백개에서 수천개에 이르는 프로세서들로 구성되어 있으나 개개 프로세서의 계산능력은 매우 약하다. 이에 반해 대분된(coarse grained) 병렬시스템은 10개미만의 강력한 처리능력을 갖는 프로세서들로 구성된다. 두번째 기준으로는 각 프로세서들의 메모리와 그를 통한 프로세서들간의 연결상태를 들수있는데 이에 따라 이미

서두에서 언급한 바와 같이 공유메모리(shared memory, paracomputer)와 분산메모리 (distributed memory, ultracomputer) 및 이 두형태의 혼합방식으로 분류된다. 공유메모리 시스템에서는 각 프로세서들이 공유하고 있는 단일의 중앙메모리에 읽고 씀으로써 그들간의 신속한 정보공유 및 교환이 가능하지만 연결된 프로세서의 수가 증가함에 따라서 병목현상을 초래한다. 분산메모리 시스템에서는 분산된 다수의 메모리들을 연결하는 통신망의 복잡성때문에 정보교환에 비교적 많은 시간이 소요된다. 프로세서들간의 통신망은 기하학적으로 규칙적인 형태를 갖는것이 보통인데 이는 일반적으로 프로세서를 나타내는 마디(vertex)와 프로세서간의 연결을 나타내는 아크(arc)로 표현된다. 각 통신망의 형태에 따른 대표적인 효율성 평가기준으로 최대 연결마디수(maximum vertex degree)와 최장통로길이(maximum path length, diameter)를 들 수 있는데 전자는 임의의 한 프로세서에 직접 연결되는 프로세서의 최대갯수를 말하는것으로 실제 제작의 타당성 관점에서 볼때 상한값을 갖게되며 후자는 임의의 두 프로세서간의 통화통로를 구성하는 아크의 최대갯수를 말하는 것으로 통화속도의 척도가된다. 다른 평가기준으로, 임의의 두 프로세서를 연결하는 독립적인 통로의 갯수를 나타내는 연결성(connectivity)은 통신망의 일부가 훼손되었거나 병목현상을 보일때 통신의 신뢰도를 나타내는 척도이며 다른 형태의 망으로 전이(mapping)될수있는 정도를 나타내는 유연도(flexibility)는 각기 다른 종류의 병렬알고리즘에 대한 시스템의 적합도를 말해준다. 세분된 통신망 형태들과 그에 따른 효율성 척도에 대해서는 Kindervater and Lenstra[17]와 Bertsekas and Tsitsiklis[3]를 참조하기 바란다.

병렬시스템에 있어서는 직렬시스템과 비교할때 분산처리에의한 연산속도의 향상이 있는 반면에 각 프로세서들간의 통화 및 자료전송으로 야기되는 피할 수 없는 계산시간의 지연도 동시에 수반한다. 물론 이러한 통화지연(communication delay)의 정도는 프로세서들이 연결된 접속망의 기하학적인 형태나 접속방법에 의하여 큰 영향을 받게된다. 총통화지연의 구성요소들을 세분하면 다음과 같다.

1) 통화처리시간(communication processing time) : 전송정보의 준비시간을 말하 는데 예를 들어 정보를 전송패킷으로 재구성하고 이에 주소 및 제어정보들을 붙이는 일이나 각 패킷의 전송경로를 결정하고 적절한 버퍼에 옮기는등의 전송에 필요한 사전절차를 수행하는데 드는 시간을 말한다.
2) 대기시간(queuing time) : 사용하고자하는 특정 전송경로의 일부분이 이미 점유되어 있거나 앞서 전송을 기다리는 패킷들 때문에 버퍼에서 대기 지연되는 시간을 말한다.
3) 전송시간(transmission time) : 패킷의 모든 비트들이 전송되는데 드는 실제 시간을 말한다.
4) 전파시간(propagation time) : 보내는 프로세서에서 최종비트의 출발시간과 받아들이는 프로세서에서 최종비트의 도착시간간의 시간적 간격을 말한다.

세분된 통화지연시간의 요소들은 주어진 시스템이나 알고리즘의 특성에 따라서 그 구성비율을 달리하는데 예를 들어 전송망들이 효율적으로 구성되어 두 프로세서간에 전송경로가 다

양하거나 전송정보들의 크기가 고르게 분포할 때는 상대적으로 대기시간이 매우 짧으며 두 프로세서간의 물리적인 거리가 가까울 경우에는 전파시간이 무시될수도있다. 대부분의 시스템에서 통화처리시간과 전파시간은 패킷마다 거의 일정하며 전송시간은 패킷에 포함된 비트수에 비례하는 것이 보통이다. 대부분 기존의 병렬처리시스템에서는 통화지연시간이 기본적인 소숫점 연산처리시간과 비교할때 상대적으로 길기때문에 직렬알고리즘을 너무 잘게(fine grained) 병렬화시켜 통화횟수를 증가시키는 경우에 병렬처리에 의한 가속효과를 감소시키는 결과를 초래할 수 도있다. 한편 공유메모리 형태의 시스템구조(paracomputer)하에서는 중앙메모리에 읽고 씀으로써 정보교환이 가능하기 때문에 전송경로의 선정이 문제되지 않으나 수천개의 프로세서로 구성된 분산 메모리시스템(ultracomputer)에서는 프로세서간의 효율적인 경로선정이 전체 병렬처리시간에 상당한 영향을 미치게된다. 경로선정문제는 임의의 프로세서 i와 j를 연결하는 링크 (i,j)에 대하여 통화지연시간의 총합을 최소화하기 위한 최단경로(shortest path)문제로 귀결되나 임의의 링크에 과다한 전송량이 부과될 경우에는 대기시간의 증가로 인한 통화체증현상이 발생한다. 이를 피하기위해 무작위 다경로선정(multi-path and randomized routing)방법에서는 출발지 A와 목적지 B사이에 매 패킷마다 서로다른 중간마디 C를 무작위로 설정하고 A와 C간의 최단경로와 C와 B간의 최단경로를 연결하여 A와 B사이의 전송경로를 결정함으로써 패킷전송을 분산시킨다. 이와같이 분산메모리 시스템에서는 시스템을 구성하는 메모리의 크기 및 구조나 알고리즘의 특성에 맞는 고유의 경로선정 알고리즘고안이 필수적이다.

이상에서 언급한 통화지연시간에 영향을 미치는 중요한 요소들을 구분, 정리하면 첫째로 전송경로선정 및 정보흐름등과 같은 통신망을 제어하는 시스템 알고리즘, 둘째로 프로세서와 메모리의 갯수, 형태 및 그들의 위치등과 같은 시스템내 통신망의 기하학적 구조, 세째로 병렬처리의 대상이 되는 문제의 구조 및 고안된 알고리즘에서 요구되는 동시화 및 병렬화의 정도를 들수있다.

3.2 알고리즘의 병렬화와 병렬처리 효과의 분석

병렬알고리즘은 유향 무순환 그래프(directed acyclic graph, DAG)로 표현될 수 있는데 G=(N,A)가 DAG라고 할때 N은 마디들의 집합이고 A는 유향아크(directed arc)들의 집합이다. 마디와 유향아크는 각각 알고리즘에 의하여 행해지는 연산과 데이타의 연관성을 나타낸다. 즉 유향아크 (i,j) A 는 마디 j에서의 연산을 위하여 마디 i의 계산결과를 사용함을 의미한다. 병렬알고리즘의 완전한 표현을 위해서는 DAG와 함께 어떤 프로세서가 언제 어떤 마디에 해당하는 연산을 행하는가 하는 구체적인 스케쥴이 수립되어야 하는데 이는 {(i,P,c) | i∈N}로 표현되며 여기서 P와 c는 각각 마디 i의 연산을 담당하는 프로세서와 그 연산의 완료시간을 말한다. 일반적으로 병렬화의 관점에서볼때 최소의 연산량을 요구하는 최적알고리즘을 찾는 것은 불가능하지만 몇개의 잘 알려진 문제들에 대해서는 이것이 가능하다.

병렬컴퓨터는 원문제로부터 분해된 독립적인 부작업(subtask)들을 여러 프로세서들에 할당하여 동시에 처리하고 나머지 종속적인 작업들에 대해서는 단일 프로세서로 직렬처리한다.

직렬처리를 위한 단일프로세서 스칼라 컴퓨터(uniprocessor scalar computer)에 비해서 CRAY X-MP, CRAY-2, IBM 3090-600, ETA-10, Alliant FX 기종등과 같이 슈퍼컴퓨터로 불려지는 다중프로세서 벡터컴퓨터(multiprocessor vector computer)는 기존의 병렬처리 기능외에 각 프로세서에 벡터 연산처리기능을 추가함으로써 병렬개념(parallelism)을 확장하였다. 작업의 병렬화는 단일 명령문에서 각 사용자의 개별작업까지 부작업(subtask)들의 크기에 따라서 여러단계로 구분가능하나 연산처리 효율의 우월성에서 보면 크게 다음의 세가지로 분류될 수 있다.

1) 다중작업처리(multitasking) : 단일 프로그램을 동시에(concurrently) 처리될 수 있는 두개 이상의 모듈들로 재구성하는 것을 말하며 통상 서브루틴형태로 짜여진 모듈들이 병렬시스템 라이브러리(parallel system library)를 통하여 불리워짐(call)으로써 연산과 저장의 독립성을 갖는 부문제들로 분리, 처리된다.
2) 미세작업처리(microtasking) : 사용자가 컴파일러 지시자(compiler directive)를 이용하여 임의로 부작업들을 구분, 처리하는것을 말하는데 일반적으로 다중작업처리로 구분된 서브루틴내의 병렬화나 DO-Loop식의 반복계산을 분산처리한다.
3) 벡터화(vectorization) : 가장 미세한 수준의 병렬화로써 DO-Loop의 가장 안쪽 반복계산들을 시스템 하드웨어로 벡터화시켜 스칼라연산과 같은 시간안에 일괄처리한다. 예를 들어 n번의 반복계산은 m개의 벡터등록자(vector registor)를 갖는 시스템을 이용하면 (n/m)차원의 벡터로 일괄처리가 가능하다.

병렬알고리즘에서는 각 프로세서가 수행하는 처리작업들에 대하여 어느정도의 중앙제어 및 관리가 행해지는것이 보통이다. 이러한 제어 및 관리는 대상의 알고리즘을 국면(phase)들로 구분함으로써 이루어지며 임의의 국면내에서는 선행국면의 결과를 토대로 여러 프로세서들이 그들간의 상호작용 없이 독립적으로 연산처리를 행하고 그 국면의 마지막 단계에서 결과들을 취합 조정하기전까지 새로운 국면으로의 진행이 지연되는등 시간적으로 제어를 받게 되는데 이를 알고리즘의 동시화(synchronization)라고 한다. 경우에 따라서는 매 국면마다 이와 같은 시간적 제어가 필요하지 않거나 상대적으로 엄격하지 않기때문에 프로세서들이 임의의 시점에 대기하여 중앙의 동시자(synchronizer)로부터 지시를 기다리지 않고 독자적으로 다음 국면으로 연산을 진행함으로써 동화지연이나 대기지연을 피할 수 있는 알고리즘의 비동시화(asynchronization)도 생각할수 있는데 이는 기본적으로 병렬화의 대상이 되는 알고리즘의 속성에 달려있다. 동시적 알고리즘에서는 각 프로세서들이 사전에 정해진 동시화 시점(synchronization point)에 대기하여 다른 프로세서들로부터 필요한 계산결과를 기다리게 되는데 이는 통상 매 국면의 마지막 시점과 일치하며 내재적으로 비동시적 구조를 갖는 알고리즘을 동시화할 경우에는 동시화 시점들을 체크하는 동시자와 같이 바람직하지 않은 프로그램부하가 발생하기도 한다. 일반적으로 동시적 알고리즘에서는 프로세서들의 관리 및 알고리즘의 단순화가 용이하다. 따라서 병렬화를 위해서는 사전에 사용할 병렬시스템의 종류, 통신망의 구조와 더불어 실행대상이 되는 알고리즘의 내재적 특성을 파악하는 것이 필요하다. 같은 자료에 대해서 매 실행마다 같은 결과를

얻는 알고리즘을 확정적(deterministic) 알고리즘이라고 하는데 동시적 알고리즘은 항상 확정적이지만 공유메모리를 갖는 통신망을 통하여 동시자 없이 여러 프로세서들이 개별적으로 읽고 쓰기를 하는 비동시적 알고리즘은 비확정적 결과를 초래할수도 있다.

병렬처리에 의한 계산상의 효과를 측정하는 척도로는 프로세서의 갯수 p의 함수인 가속성(speedup) S_p와 효율성(efficiency) E_p를 들수 있다. 가속성은 병렬화에 의한 총 실행시간의 단축을 나타내는 것으로 이는 단일 프로세서에 의한 최적(optimal) 직렬알고리즘의 실행시간을 병렬컴퓨터에 의한 병렬알고리즘의 실행시간으로 나눈 값이다. 효율성은 이상적인 가속성과 비교한 실제 가속성의 질을 말해주며 실제 가속성을 프로세서의 갯수로 나누어서 구한다. 이론적으로 가능한 가속성과 효율성의 값은 각각 p와 1이지만 통화지연이나 동시화에 따른 여러가지 부하들로 인하여 이론치에 미치지 못하는 것이 보통이다. 실제로 가속성을 계산하기 위하여 필요한 최적 직렬알고리즘의 실행시간은 이론치에 불과한 것으로써 임의의 직렬알고리즘이 주어진 문제에 대하여 최적해법이라는 것을 증명하는 것은 불가능하다. 따라서 이 이론적인 시간에 대한 몇가지 대안들로 첫째로 기존하는 최상(best)의 직렬알고리즘의 실행시간, 둘째로 컴퓨터 수행속도의 비교기준으로 사용되는 벤취마크 직렬알고리즘의 실행시간, 셋째로 분석대상이되는 병렬알고리즘을 단일 프로세서에 의하여 실행했을 때의 수행시간등을 들수있다. 본 연구에서는 마지막 대안에 의한 효율성을 측정기준으로 삼았는데 이는 특정의 알고리즘이 얼마나 잘 병렬화되었는지를 설명할 뿐 그 병렬알고리즘 자체의 절대적 효율을 설명하지는 못한다. 대부분의 병렬화 예에서 보는 바와 같이 어떤 부분은 병렬화가 가능하지만 다른 부분은 내재적으로 직렬처리 구조를 갖기때문에 완전 병렬화가 어렵다. 이러한 부분적인 병렬화에 의한 가속성에 상한값을 제시하는 Amdahl의 법칙[2]은 다음과 같다.

$$S_p \leq \frac{1}{d+(1-d)/p}\ ,\ \forall p$$

여기서 d는 총 실행시간에 대한 직렬처리시간의 비율이며 이는 작은 d값에 의해서도 가속성이 크게 감소함을 간접적으로 보여준다.

4. Lagrangean 근사과정의 병렬화

4.1 직렬 Lagrangean 근사과정 (LAP)

Lagrangean 근사과정(LAP)은 임의의 조합적(combinatorial) 최적화모형에 대하여 제약식의 일부나 복사제약식(copy constraint)를 완화한 Lagrangean 완화문제를 최소(최대)화하고 동시에 이 목적값을 최대(최소)화하는 Lagrangean 승수를 구함으로써 원 문제의 하한(상한)을 제시해 주는 한정(bounding)기법이다. 즉 주어진 승수 v에 대해서 원 문제 (P)의 Lagrangean 완화문제를 LR(v)라고 하면 Lagrangean 쌍대문제 LD는 $\max_v$ LR(v)로, 혹은 완화문제가 부문제 $LR_i(v)$들로 분해될 경우에는 $\max_v \sum_i LR_i(v)$로 정의되며 OV(LD)≤OV(P)의 관계를 갖는다. Lagrangean

완화문제의 목적값으로 이루어진 Lagrangean 쌍대함수(dual function)는 승수 v에 대해 연속적이며 구분적 선형(piecewise linear)인 볼록(convex)함수임이 알려져있다[11]. 구분적 선형인 볼록함수의 정점을 찾기위한 등정법(hill climbing method)으로 여러가지가 있으나 본 연구에서는 subgradient법에 의한 승수 수정을 가정한다. subgradient 법에서는 임의의 Lagrangean 승수에 대한 완화문제 또는 분해된 부문제들의 최적해로부터 원문제의 하한을 결정함과 동시에 그 점에서 Lagrangean 함수의 subgradient를 계산하고 이 방향으로 정해진 보폭에 따라 승수를 조정하여 다음 계산단계에 수정된 승수를 제공하는 등 반복적인 계산을 계속하게 된다. 이러한 반복계산의 종료조건으로는 근사정도, 보폭의 감소추세 또는 계산반복횟수등을 들수있다. 때에 따라서는 subgradient법의 매 단계에서 휴리스틱에 의해 원문제의 가능해를 도출해냄으로써 원 목적함수값에 대한 상한(하한)을 동시에 제공하기도 한다.

LAP를 네 단계로 구분 정리하면 다음과 같다. 첫째로 완화 및 분해(relaxation and decomposition)단계에서 원문제의 제약식 일부가 완화되고 이로 인하여 완화문제가 소규모의 독립적인 부문제들로 분해된다. 둘째로 부문제 최적화 및 하한(상한)의 갱신(solving subproblems and updating lower bound)단계에서는 주어진 승수에 대해 부문제들의 최적해가 구해지고 이로부터 원목적함수값에 대한 하한이 갱신된다. 셋째로 실행종료검사(termination checking)단계는 사전에 정해진 종료조건과 비교하여 부합하면 알고리즘의 실행을 종료한다. 마지막으로 승수갱신(multiplier adjustment) 단계에서 subgradient가 계산되며 이에 따라 현재의 Lagrangean 승수를 갱신한다.

4.2 병렬 Lagrangean 근사과정 (PLAP)

LAP는 두가지 형태의 반복적(iterative) 알고리즘으로 쉽게 기술된다. 그 첫번째는 Lagrangean 승수 v를 통한 반복적 알고리즘으로

$$v(k+1)=f(v(k)), \qquad k=0,1,2,\cdots\cdots,K$$

의 형태로 주어지며 여기서 $K=\arg\max_k\{H(k,\lambda,LB,UB)\le\varepsilon\}$ 로써 반복계산의 종료횟수를 말하고 v(0)는 Lagrangean 승수의 초기치이다. Lagrangean 승수벡터의 순열(sequence) $\{v(k)\}$는 반복계산의 실행종료조건 $H(k,\lambda,LB,UB)\le\varepsilon$를 만족할때까지, 부문제 최적화 및 하한의 갱신단계와 승수갱신단계에서 행해지는 연산들을 표현하는 함수 f(·)에 의하여 발생된다. 계산단계 k에서 주어진 승수 v(k)에 대하여 Lagrangean 완화문제들이 최적화되며 그들의 최적해들로부터 계산되어진 subgradient에 의하여 갱신되어진 승수 v(k+1)가 다음 계산단계 (k+1)로 넘겨지게된다. 보폭 에 대한 일정의 조건하에서 $\{v(k)\}$가 수렴한다는 사실이 이미 알려져 있다[15].

두번째 반복적 알고리즘은 주어진 승수 v에 대한 Lagrangean 완화문제의 최적해 $x\in OS(LR(v))$의 함수형태인

$$x(k+1)=g(x(k)), \qquad k=1,2,\cdots\cdots,K \qquad (1)$$

로 표현되며 $x(k)\in OS(LR(v(k-1)))$ 이다. $\{x(k)\}$는 LAP가 종료할때까지 함수 g(·)에 의하여 발생되는 순열이다. $x(k)\in OS(LR(v$

(k−1)))와 $g_i(\cdot)$를 각각 x(k)와 $g(\cdot)$의 i번째 요소라고하면 (1)은

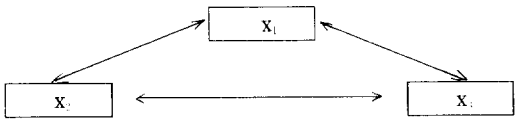
$$x_i(k+1)=g_i(x_1(k),x_2(k),\cdots\cdots,x_N(k)),$$
$$i\in I, \quad k=1,2,\cdots,K \qquad (2)$$

로 다시 쓰여질수 있으며 여기서 I는 Lagrangean 완화문제들의 집합이고 N은 |I| 이다. LAP의 병렬화는 각 $g_i(\cdot)$의 연산을 프로세서들에 할당함으로써 가능하다. 계산단계 k에서 i번째 프로세서는 공유메모리나 프로세서들간의 통신망을 통하여 $x_i(k)$ $\forall i\in I$ 로부터 계산되어진 v(k)를 알게되며 다시 $g_i(\cdot)$에 의하여 $x_i(k+1)\in OS(LR_i(v(k)))$를 계산하고 그 결과를 다음 계산단계의 승수 v(k+1)를 계산하기 위하여 공유메모리에 기록한다.

이와같이 매 계산단계내의 총 계산량이 다수의 프로세서에 분배되는데 병렬화의 효율면에서 볼때 중요한 것은 첫째로 프로세서들에 계산량이 균등하게 분배되었는가 하는것이고 둘째로 각 프로세서에서 독립적으로 처리될 수 없는 직렬처리부분이 얼마나 되는가 하는것이다. 매 계산단계에서 임의의 단일 프로세서들이 독립적으로 수행하는 일은 크게 세가지로 구분되는데 그 첫째가 전 단계의 결과로부터 갱신된 승수를 공유메모리로부터 읽어들이는 일이고 두번째로 그 승수에 기초하여 자기에게 할당된 Lagrangean 완화문제를 최적화하는 일이며 셋째로 그 결과로 얻어진 최적해를 subgradient의 계산을 위하여 공유메모리에 기록하는 일이다. 매 계산단계의 마지막 시점에서 각 프로세서들로부터 받는 부문제들의 최적해들을 취합하고 이들로부터 Lagrangean 쌍대함수의 subgradient를 계산하고 이에 따라 승수를 갱신하는 일이 PLAP내에서 직렬로 처리되는 부분이다. 단일 프로세서에 의해서 독립적으로 처리되는 계산량중 대부분을 차지하는 것은 할당된 완화문제들의 최적화로써 앞서 살펴본 분해예에서와 같이 동일유형의 부문제들로 분해되는 생산문제인 경우에는 최악의 문제자료(worst case problem instance)를 갖는 특정한 부문제의 최적화 과정을 제외하면 프로세서들간에 계산량이 균등히 분배됨을 예상할수 있다. 단지 부문제의 갯수가 프로세서수의 배수가 아닌 경우에 발생할 수 있는 일부 프로세서들의 유휴시간이 알고리즘 동시화에 약간의 지연을 초래할 수도 있다. Lagrangean 쌍대함수의 subgradient를 계산하는 직렬처리부분은 전체 계산량에 비교해볼때 매우 작을뿐만아니라 벡터계산에 적합한 반복적 계산구조를 갖기 때문에 병렬화에 의한 계산의 가속성에 크게 영향을 미치지 않는다. 매 계산단계는 동시화 시점으로 구분되어 일부 프로세서가 자신들에게 할당되어진 계산을 종료했음에도 불구하고 나머지 다른프로세서들의 계산 및 그 결과의 통신이 완료되기 전에는 다음 계산단계로 독자적으로 진행하는것이 불가능하다. 이러한 동시화 지연(synchronization delay)을 최소화하기 위해서는 프로세서들로의 작업량 분배에 관한 효율적인 계획이 필수적이다.

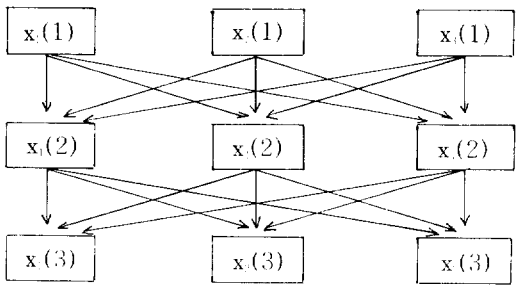
(1)에서 보는바와 같이 주어진 v(k)에 대하여 x(k+1)은 x(k)에 종속적이므로 LAP는 수렴속도가 빠른 Gauss-Seidel 식의 병렬화가 불가능하다. Gauss-Seidel 병렬화란 벡터 x(k+1)의 각 요소들이 가장 최근에 갱신된 정보의 일부를 우선적으로 이용하여 계산되는 방식을 말하는 것으로 $x_i(k+1)=g_i(x_1(k+1),x_2(k+1),\cdots,x_{i-1}(k+1), x_i(k),\cdots,x_N(k))$ 로 표현된다. 예를 들어 N=3 일때 (2)의 반복계산을 수행하는데 필요한 정보교환을 표시하는 LAP의

연관그레프(dependency graph)는 [그림 1]과 같으며 여기서 마디는 벡터 x의 요소를, 유향 아크는 두 마디간의 정보교환을 나타낸다.



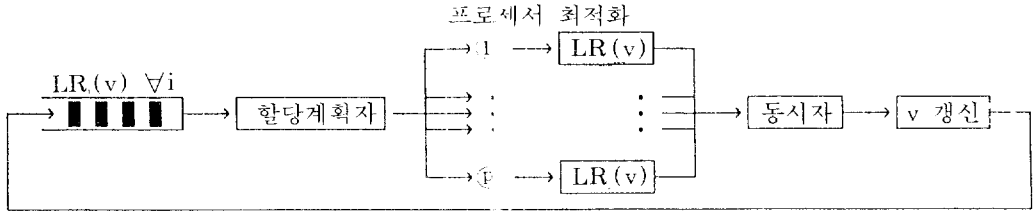
[그림 1] N=3 일때 LAP의 연관그래프

반복계산 (2)가 k=1,…,K 에 대해서 행해질 때 PLAP의 구조는 위의 연관그레프가 시간상 전개되는 모양을 갖는 DAG로 표현된다. [그림 2]는 x(k+1)의 모든 요소들이 동일시점, 즉 Step2 : 부문제 최적화 및 하한의 갱신단계에서 동시에 갱신되는 동시 Jacobi식 PLAP를 K=3 일때 DAG로 표현한 것이다.



[그림 2] K=3 일때 동시 Jacobi식 PLAP를 표현한 DAG

동시 Jacobi식 PLAP에서 각 프로세서는 $x_i(k+1) \in OS(LR_i(v(k)))$를 구하는데 배정되며 이들로부터 얻어진 부문제들의 해 $x_i(k+1)$ $\forall i$은 subgradient $\pi(v(k))$를 계산하기 위하여 통신망을 통하여 동시에 전달되거나 공유메모리에 기록된다. 분해된 부문제들의 갯수 N이 프로세서의 갯수 p보다 큰 경우에는 일부의 프로세서들은 한개 이상의 부문제들을 풀게되며 반대로 N < p 이면 (p−N)개의 프로세서는 부문제 최적화 단계에서 처리작업 없이 놀게됨으로써 병렬화의 효율성을 떨어뜨리는 결과를 초래하게된다. 이상의 두가지 어느 경우라도 부문제 최적화 단계에서 동시에 행해지는 연산작업의 수는 프로세서의수와 처리를 기다리는 미처리 부문제의 갯수중 그 최소치와 일치한다. 이와같이 시간이 흐름에 따라서 미처리 부문제의 갯수와 문제자료(problem instance)가 변하기 때문에 병렬작업의 부하는 매 계산단계마다 변하게된다. 미처리 부문제의 갯수가 가용한 프로세서의 갯수보다 작음으로써 발생되는 유휴시간을 최소화하기 위해서는 매 계산단계마다 병렬처리되는 부문제 최적화 단계의 기간(makespan)을 최소화하는 프로세서로의 부문제 할당계획이 선행되어야 함은 물론이다. [그림 3]은 동시 Jacobi식 PLAP의 구조를 보여준다.



[그림 3] 동시 Jacobi식 PLAP의 구조

매 계산단계마다 하나의 동시화 시점이 포함되는데 이는 모든 부문제들 $LR_i(v)$ $\forall i$가 최적화되어 그들의 최적해들이 취합될때까지 Step 4 : 승수갱신단계의 시작을 지연한다. 할당계획자(scheduler)는 병렬화된 부문제 최적화 단계의 기간을 최소화하도록 p개의 프로세서들에 부문제들을 할당하는데 이에 따라 PLAP의 효율이 크게 영향을 받게된다. 그러나 이 할당계획자에 의하여 행해지는 최적의 스케쥴링은 바로 NP-hard문제로 알려진[19] 기간 최소화를 위한 비단속 병렬 스케쥴링 문제(nonpreemptive parallel scheduling problem with minimizing makespan)이다. 본 연구에서는 두가지의 휴리스틱 할당계획자를 제시하는바 그 하나가 최장공정시간 규칙(longest processing time rule : LPT rule)이다. LPT 규칙은 매 계산단계마다 모든 부문제들을 처리시간에따라 내림차순으로 순서를 매긴후 이들을 순서대로 가장적은 계산부하량을 갖는 프로세서에 우선적으로 할당하는 방식이다. 다른 또하나의 규칙은 LPT 규칙과는 달리 사전에 부문제의 처리시간들을 모를때 - 실제로 부문제들이 풀어지기 전까지는 그들의 처리시간을 모르는 경우가 일반적이다 - 사전에 임의로 정해놓은 순서목록(list)에 따라 미처리된 부문제들을 연속적으로 프로세서를 관찰하여 유휴상태에 있는 프로세서에 순차적으로 할당하는 순서목록 스케쥴링 규칙(list scheduling rule : LS rule)이다. 병렬처리 대상이 되는 부문제들이 동일한 유형의 최적화 문제들로써 처리시간이 비슷할것으로 예상되는 경우에는 무작위의 처리순서를 정하는것도 무방하나 여러가지 서로 다른 유형의 부문제그룹들로 분해되어 그들의 처리시간이 서로 상이하다고 판단될때는 동일 그룹안의 처리순서는 무작위로 하되 그룹별 처리순서는 LPT 규칙을 따르는 것이 현명하다. 이 두가지 휴리스틱 할당계획자들에 대한 최악의 경우에 있어서의 수행도 평가(worst-case performance)는 Graham[12,13]의 연구결과를 이용하여 다음과 같이 주어진다.

[정리 1] 프로세서가 p개일때 LPT 규칙과 LS 규칙에 의한 처리기간(makespan) $T_p(LPT)$와 $T_p(LS)$의 최악의 한계(worst-case bound)는 각각

$$T_p(LPT) \;/\; T_p^* \le 4/3-1/3p$$
$$T_p(LS) \;/\; T_p^* \le 2-1/p$$

로 주어지며 여기서 T_p^*는 p개의 프로세서로 처리한 부문제 최적화 단계의 이론적인 최적처리기간이다.

t_{ik}를 계산단계 k에서 부문제 $LR_i(v)$의 처리시간이라 하고 $T_p^k(LS)$를 LS 규칙을 사용했을때 계산단계 k에서 부문제 최적화 단계의 처리기간(makespan)이라고 하면 LS규칙에 의한 PLAP의 효율성 $E_p(LS)$는 Amdahl의 법칙으로부터 다음과 같이 계산된다.

$$E_p(LS)=\frac{\sum_{k=1}^{K}\sum_{i=1}^{N} t_{ik}+(K-1)\delta}{p\left[\sum_{k=1}^{K} T_p^k(LS)+(K-1)\,\delta\right]}$$

여기서 δ는 승수갱신단계의 처리시간으로서 t_{ik}에 비하면 매우 짧고 매 계산단계마다 일정한 값을 갖는다. t_{ik}의 평균값과 분산이 작을수록 각 프로세서의 유휴시간이 감소하므로 PLAP의 효율성은 높아진다.

5. Alliant FX/4를 이용한 Lagrangean 근사과정의 병렬처리 및 계산결과

Alliant FX/4 는 연산요소(computational element : CE) 라고 불리어지며 주로 고속의 계산을 전담하는 4개의 프로세서와 입출력과 시스템의 운용체계를 담당하는 6개의 상호작용 프로세서(interactive processor : IP)들로 구성되어진 병렬컴퓨터이다. CE들및 IP들과 64M 바이트의 공유메모리는 공유메모리 버스를 통하여 연결되어 있으며 이들 각 프로세서는 128K 바이트의 캐쉬메모리를 통하여 버스에 접근한다. 또한 CE들끼리는 동시제어버스(concurrency control bus)를 통하여 직접 연결되어 있으며 각 CE는 MC68000 구조와 호환성을 갖으며 소숫점연산, 백터연산 및 동시계산기능이 보강된 M68020 연산셋을 지원하는 주문칩(custom chip)을 사용하고있다. 백터계산모드에서 각 CE는 단정도(single precision)의 소숫점계산을 최고 11. 8 Mflops의 초고속으로 수행하며 따라서 총 4개의 프로세서에 의한 병렬처리속도는 47 Mflops에까지 이른다. 다수의 CE들이 동일한 프로그램내 임의의 loop에 포함된 반복계산들을 나누어서 동시계산할 수도 있는데 시스템 하드웨어가 반복계산들을 가용한 프로세서에 자동적으로 분배해 주기때문에 이를 self-scheduled do-across loop라고 부르며 이러한 loop의 처리가속성은 최고 CE의 갯수와 같다. 이와 더불어 각 CE의 통관(pipelining)능력은 사칙연산들을 겹쳐 진행시킴으로써 동일 처리자료들에 대한 복수연산들간에 시간적인 단절로 인한 처리지연을 없애며 보다 효과적인 가속을 위해서는 처리대상이 되는 자료가 큰 규모의 백터형태이어야 함은 물론이다. 본 연구에 사용된 병렬컴퓨터 Alliant FX/4는 미국 Pennsylvania대학 부설 병원(HUP)내 화상처리(image processing) 연구실에 위치하고 있으며 국내에서는 telnet 프로토콜을 이용하여 국제학술 통신망중에 하나인 internet상에서 mipgsun.mipg.upenn.edu (128.91.14.150)으로 원격접근(remote access)함으로써 그 사용이 가능하다. 사용자는 각종 Alliant 프로그래밍언어의 컴파일러 지시자(compiler directive)를 이용하여 네가지 실행모드 - 스칼라(scalar : S), 백터(vector : V), 동시(concurrent : C), 백터-동시(vector-concurrent : V-C) - 들을 병렬화의 대상이되는 원(source) 프로그램내에 적절히 삽입, 구성하여 최적의 병렬연산을 실행한다.

Alliant FX/4상에서 병렬실행의 예로 든 다단계 조립공정에서의 롯사이징 문제(lot-sizing problem in multistage assembly systems) [1]에 대한 LAP의 구조는 다음과 같이 주어진다.

Step 1 (초기화)
조립공정의 구조입력
LB 와 UB 의 초기치 설정

Step 2 (원문제의 완화 및 분해)
(LR4(v))가 각 품목 i에 대해서 단계 롯사이징 문제 (LR4$_i$(v))들로 분해됨

Step 3 (부문제의 풀이와 하한치의 갱신)
DO $\forall i \in I$
Wagner-Whitin의 알고리즘에 의하여 (LR4$_i$(v))를 푼다
ENDDO
LB $\leftarrow$ max {LB, Σ_i OV(LR4$_i$(v))}

Step 4 (Lagrangean 휴리스틱에 의한 상한치

〈표 1〉 직렬프로그램의 요약된 프로파일 (반복계산횟수 : 301번 기준)

부작업	실행시간구성비(%)	누적시간(초)	실행시간(초)	서브루틴콜횟수
Step1과 2	0.3	0.29	0.29	1
Step3	72.9	73.00	74.71	15050
Step4	19.6	93.10	20.10	1
Step5	7.2	100.48	7.38	1

의 갱신)

UB ← min {UB, UBB(v)}

UBB(v) : 주어진 v에서 Lagrangean 휴리스틱에 의한 상한치

Step 5 (실행종료검사 및 승수갱신)

만약 $H(k,\lambda,LB,UB) \leq \varepsilon$ 이면 실행종료

$v \leftarrow \max\{v + \lambda \cdot \pi(v), 0\}$

Step 3 실행

계산 효율의 비교를 위하여 공정 단계수(number of levels)에 따라 A, B, C 세가지 문제유형[1]으로 구분하였으며 각각 계획기간 12, 5개에서 10개 사이의 개별공정수(number of nodes)를 갖는 90개의 무작위 발생문제들을 근거로 PLAP의 효율을 검증하였다.

본래 Alliant FORTRAN의 자동 컴파일러 최적화 기능(automatic compiler optimization)은 사용자의 특별한 지시없이도 자동적으로 원 프로그램을 일괄적으로 병렬화시키는등 사용상 매우 편리하지만 일반적으로 원 프로그램의 특별한 구조에 따른 부분적 병렬화를 허용하지 않기때문에 사용자 개인적 수작업에 의한 최적화(customized manual optimization)와 비교해 볼때 그 효율이 많이 떨어진다. 원 프로그램을 수작업으로 최적 컴파일하기 위해서는 우선 프로그램내 각 부분들의 직렬실행시간 또는 DO loop나 서브루틴의 위치, 상호관계등 원 프로그램의 상세한 구조를 알려주는 프로파일 리스트(profile list)가 필요하다. 예를 들어 50개의 개별 공정수를 갖는 C형 다단계 조립공정의 롯사이징 문제를 위한 직렬프로그램의 요약된 프로파일은 〈표 1〉과 같다.

전체시간의 0.3%를 차지하는 Step1과 2의 실행시간 대부분은 초기 가능해를 찾는데 드는 것으로 이는 원재료부터 시작해서 공정순서를 따라 완제품의 생산방향으로 행해지는 계산적으로 의존적인 방식으로 구해지므로 벡터-동시(V-C) 실행모드로 완전 병렬화하는 것은 불가능하다. 이에 반하여 Step3는 전체 실행시간의 대부분(72.9%)을 차지하여 병렬화의 주요 대상이 되는데 Wagner-Whitin(W-W) 알고리즘으로 구성된 각 서브루틴은 0.00496초 동안 실행되고 매 반복계산 단계마다 개별공정수와 같은 50번씩 독립적으로 행해진다. Step3의 제일 바깥쪽 루프는 동시(C)모드로 실행하여 W-W 서브루틴들은 무작위 처리순서를 갖는 LS규칙에 따라 각 프로세서에 할당되어 병렬계산 되어진다. 또한 계산처리의 가속화를 위하여 각 W-W는 벡터(V)모드로 세분 병렬화된다. Step4는 Lagrangean 휴리스틱으로써 계산적으로 의존적인 상하(top-down) 방식이므로 V-C 모드의 실행이 불가능하지만 승수갱

신절차 Step5는 V-C모드로 완전 병렬화가 가능하다. 전체 실행시간의 80.1%(Step3과 Step5)가 완전병렬화 가능하므로 Amdahl의 법칙에 의한 가속성에 대한 상한치는 각 각 1. 67(p=2), 2. 14(p=3), 2. 50(p=3) 이다.

〈표 2〉와 〈표 3〉은 각 각 네가지 실행모드 즉 S, 자동컴파일러최적화(AUTO), C, V-C와 세가지 문제 유형에 대하여 병렬프로그램의 실행시간과 가속성을 정리해 놓은 것이다. AUTO실행모드에서는 전술한바와 같이 컴파일러 지시자를 원 프로그램의 적절한 위치에 에 삽입하여 사용자의 의도대로(customized) 병렬화하지 않고 컴파일러가 자동으로 프로그램의 구조를 인식하여 임의로 병렬계산 되어지는데 이는 수작업에 의한 병렬화와는 달리 W-W 알고리즘 내부에서 부터 필요이상의 세분된 완전병렬화를 함으로써 Alliant FX와 같은 공유메모리 시스템상에서는 프로세서간에 과도한 통신부담을 야기하여 결과적으로 계산의 가속성을 저하시킨다. 이에 반해 수작업 병렬화는 사용자의 판단에 의해 여러가지 컴파일러 지시자를 적절한 위치에 사용하여 효율적으로 병렬화 하는 방식으로 경우에 따라서는 원 프로그램을 병렬화에 맞도록 수정하거나 모듈화함으로서 자료의 의존성을 완화하고 실행 컴퓨터의 속성에 따라서 그 분할의 정도를 조절하기도 한다.

본 연구에서는 전술한 수작업 병렬화를 프로세서 네개까지 사용하며 C와 V-C의 두가지 모드로 실행하여 그 결과를 AUTO 및 S모드의 그것과 비교함으로써 병렬화의 가속성을 원인별로 규명한다. 〈표 2〉에서 문제 유형에 따른 실행시간의 차이는 뚜렷하지 않고 다만 각 문제 유형내에서 실행시간이 개별공정수에 따라 선형적으로 증가함을 알수 있다. 〈표 3〉은 수작업 병렬화에 의하여 달성된 총 가속성을 동시화와 백터화의 요인별로 구분하여 보여준다. 즉 열 (S)/(AUTO)는 자동 컴파일러 최적화에 의한 가속성을 말하며 (S)/(C), (C)/(V-C)와 (S)/(V-C)는 각 각 동시화, 백터화와 백터-동시화에 의한 가속성을 나타낸다. 네개의 프로세서를 사용할 경우 AUTO모드에 의한 가속성은 최고 1. 54인 반면 수작업 병렬화에 의한 가속성은 V-C모드로 최고 5. 97까지 기록하였다. 특히 고정의 프로세서 갯수에 대해 개별 공정수 즉 부문제 수가 20개까지 증가함에 따라 가속성이 증가함을 알수있는데 이는 각 프로세서로의 할당 대상이 되는 부문제의 갯수가 소수일때 생겨나는 분배의 불균형이 그원인으로 추측된다. 효율성은 사용되는 프로세서의 갯수가 증가함에 따라서 떨어지는 현상을 보인다. 동시화에 의한 가속성은 네개의 프로세서를 사용할 경우 최대 2. 50까지 증가하며 백터화에 의한 가속성은 프로세서의 갯수와 무관하고 최대 2. 45까지 기록하여 전체 가속성에 크게 기여함을 알수 있다.

〈표 2〉 평균 실행시간(초)

유형	크기[1]	(S)[2]	(AUTO)[3]	프로세서의갯수(p)[4]					
				2		3		4	
				(C)[5]	(V−C)[6]	(C)	(V−C)	(C)	(V−C)
A	5	1.51	1.02	1.12	0.54	0.88	0.42	0.86	0.42
	10	11.91	7.93	7.74	3.40	6.54	2.92	5.49	2.50
	20	26.83	17.58	17.19	7.23	13.39	5.68	10.93	4.76
	30	53.67	35.04	34.11	14.13	25.90	10.80	22.32	9.44
	40	67.50	43.96	42.87	17.58	33.18	13.70	27.00	11.30
	50	101.40	65.82	64.41	26.25	48.90	20.08	41.24	17.13
B	5	7.04	4.82	5.25	2.45	4.05	1.90	3.98	1.87
	10	7.90	5.24	5.17	2.27	4.49	1.95	3.69	1.67
	20	33.36	21.79	21.46	9.03	16.79	7.13	13.72	5.97
	30	53.30	35.32	34.80	14.43	26.46	11.06	22.89	9.67
	40	74.74	48.53	47.76	19.60	37.11	15.34	30.31	12.71
	50	101.79	65.77	64.96	26.58	49.61	20.42	41.96	17.46
C	5	0.59	0.40	0.45	0.21	0.35	0.16	0.34	0.16
	10	9.70	6.46	6.30	2.79	5.34	2.39	4.50	2.04
	20	35.96	23.55	23.09	9.75	18.05	7.68	14.79	6.43
	30	17.13	30.79	30.16	12.51	22.91	9.58	19.84	8.38
	40	71.75	46.65	45.80	18.85	35.52	14.70	29.05	12.16
	50	85.74	55.69	54.63	22.38	41.77	17.19	35.25	14.67

주) 표의 수치는 5문제의 평균 실행수치임

1. 개별공정수
2. 스칼라 모드(직렬연산)
3. 4개의 프로세서에의해 실행된 자동컴파일러 최적화
4. 수작업 병렬화
5. 동시 모드
6. 백터−동시 모드

〈표 3〉 동시화와 백터화의 효과

유형	크기	(S) /(AUTO)[1]	가속성(Sp)								
			(S)/(C)[2]			(S)/(V−C)[3]			(S)/(V−C)[4]		
			2	3	4	2	3	4	2	3	4
A	5	1.48	1.34	1.71	1.75	2.07	2.09	2.04	2.80	3.60	3.60
	10	1.50	1.53	1.82	2.06	2.27	2.24	2.20	3.50	4.07	4.76
	20	1.52	1.56	2.00	2.35	2.35	2.35	2.29	3.71	4.72	5.64
	30	1.53	1.57	2.07	2.40	2.40	2.40	2.36	3.80	4.97	5.69
	40	1.53	1.57	2.03	2.50	2.42	2.42	2.39	3.84	4.93	5.97
	50	1.54	1.57	2.07	2.45	2.44	2.44	2.41	3.86	5.05	5.92
B	5	1.46	1.34	1.73	1.77	2.14	2.13	2.13	2.87	3.71	3.76
	10	1.50	1.52	1.79	2.04	2.27	2.25	2.21	3.48	4.05	4.73
	20	1.53	1.55	1.98	2.43	2.38	2.35	2.30	3.69	4.68	5.59
	30	1.53	1.56	2.05	2.37	2.41	2.39	2.38	3.76	4.91	5.61
	40	1.54	1.56	2.01	2.47	2.44	2.42	2.38	3.81	4.87	5.88
	50	1.54	1.56	2.05	2.43	2.44	2.43	2.40	3.83	4.98	5.83
C	5	1.47	1.31	1.68	1.74	2.15	2.18	2.13	2.81	3.69	3.69
	10	1.50	1.53	1.81	2.16	2.26	2.23	2.21	3.47	4.06	4.75
	20	1.52	1.55	1.99	2.43	2.37	2.35	2.30	3.69	4.68	5.59
	30	1.53	1.56	2.06	2.38	2.41	2.39	2.37	3.77	4.92	5.62
	40	1.53	1.56	2.02	2.47	2.43	2.42	2.39	3.81	4.88	5.90
	50	1.54	1.57	2.05	2.43	2.44	2.43	2.40	3.83	4.99	5.84

주) 표의 수치는 5문제의 평균 실행수치임

1. 자동 컴파일러 최적화에 의한 가속성
2. 동시화에 의한 가속성
3. 백터화에 의한 가속성
4. 백터−동시화에 의한 가속성

6. 결론 및 연구방향

본 연구에서 Lagrangean 근사과정의 병렬화에 관한 이론적 지침이 정리되었고 이에 따라 예로든 다단계 롯사이징 문제에 대하여 Alliant FX/4 병렬컴퓨터를 이용한 동시 Jacobi PLAP의 계산상 가속성을 검증하였다. 보다 개선된 가속성을 갖는 비동시 PLAP를 위해서는 비동시 승수갱신의 수렴성이 우선 증명되어야 할 것이다. 계산 결과로는 수작업 병렬화의 가속성이 AUTO 모드의 그것보다 훨씬 높은 5. 97로 기록되었으며 이 총 가속성은 동시화와 백터화에 기인한것으로 각 각의 효과는 최대 2. 50과 2. 39로 조사되었다. 이는 간접적으로 Lagrangean 근사과정이 백터-동시화에 의한 병렬계산에 매우 적합함을 시사한다. 앞으로의 연구과제로 분산 메모리 컴퓨터구조에 맞는 PLAP의 고안과 실행이 필요하리라 사료된다.

참 고 문 헌

[1] Afentakis, P., B. Gavish and U. S. Karmarkar, "Computationally Efficient Optimal Solution to the Lot-sizing Problem in Multistage Assembly Systems," *Management Science*, Vol. 30(1984), pp. 222-239.

[2] Amdahl, G., "The Validity of Singlr Process Approach to Achieving Large Scale Computing Capabilities," *AFIPS Proceedings*, Vol. 30(1967), pp. 783-785.

[3] Bertsekas, D. and J. Tsitsiklis, *Parallel and Distributed Computation - Numerical Method*, Prentice Hall, New Jersey, 1989.

[4] Brown, R., J.F. Shapiro and P.J. Waterman, "Parallel Computing for Production Scheduling," *Manufacturing System*, October(1988), pp. 59-63.

[5] De Bruin, A., A. H. G. Rinnooy Kan and H. W. J. M. Trienekens, "A Simulation Tool for the Performance Evaluation of Parallel Branch and Bound Algorithm," Erasmus University, 1987, Rotterdam, The Netherlands.

[6] Dobson, G. and U. S. Karmarkar, "Simultaneous Resource Scheduling to Minimize Weighted Flow Times," *Operations Research*, Vol. 37(1989), pp. 592-600.

[7] Erlenkotter, D., "A Dual Based Procedure for Uncapacitated Facility Location," *Operations Research*, Vol. 26 (1978), pp. 992-1009.

[8] Fayard, D. and G. Plateau, "An Algorithm for the Solution of the 0-1 Knapsack Problem," *Computing*, Vol. 28(1982), pp. 269-287.

[9] Fisher, M. L. and R. Jaikumar, "A Generalized Assignment Heuristic for Vehicle Routing," *Networks*, Vol. 11 (1981), pp. 109-124.

[10] Flynn, M. J., "Very High-Speed Computing Systems," *Proc. IEEE*, Vol. 54 (1966), pp. 1901-1909.

[11] Geoffrion, A. M., "Lagrangean Relaxation for Integer Programming," *Math. Prog. Study*, Vol. 2(1974), pp. 82-113.

[12] Graham, R. L., "Bound for Certain Multiprocessing Anomalies," *The Bell System Technical Journal*, Vol. 45 (1966), pp. 1563-1581.

[13] Graham, R. L., "Bounds on Multiprocessing Time Anomalies," *SIAM J. Appl. Math.*, Vol. 17(1969), pp. 416-429.

[14] Graves, S. C., "Using Lagrangean Techniques to Solve Hierarchical Production Planning Problems," *Management Science*, Vol. 28(1982), pp. 260-275.

[15] Held, M., P. Wolfe and H. D. Crowder, "Validation of Subgradient Optimization," *Math. Prog.*, Vol. 6 (1974), pp. 62-88.

[16] Kempa, D. N., J. L. Kenington and H. A. Zaki, "Performance Characteristics of the Jacobi and Gauss-Seidel Versions of the Auction Algorithm on the Alliant FX/8," Operations Reserach Laboratory, University of Illinios at Urbana-Champaign, 1989.

[17] Kindervator, G. A. P. and J. K. Lenstra, "Parallel Computing in Combinatorial Optimization," *Annals of Operations Research*. Vol. 14(1988), pp. 245-289.

[18] Klincewicz, J. G., H. Luss and E. Rosenberg, "Optimal and Heuristic Algorithms for Multiproduct Uncapacitated Facility Location," *E. J. O. R.*, Vol. 26(1986), pp. 251-258.

[19] Lenstra, J. K., A. H. G. Rinnooy Kan and P. Brucker, "Complexity of Machine Scheduling Problems," *Annals of Discrete Mathematics*, Vol. 1(1977), pp. 343-362.

[20] Meyer R. R. and S. A. Zenios(eds.), *Parallel Optimization on Novel Computer Architectures. Annals of Operations Research*, Vol. 14(1988).

[21] Mulvey, J. M. and M. P. Beck, "Solving Capacitated Clustering Problems," Princeton University, 1982.

[22] Pirkul H., "Efficient Algorithms for the Capacitated Concentrator Location Problem," *Comput. Opns. Res.*, Vol. 14(1987), pp. 197-208.

[23] Ribeiro, C. C., "Parallel Computer Models and Combinatorial Algorithms," *Annals of Discrete Mathematics*. Vol. 5(1987), pp. 113-138.

[24] Trienekens, H. W. J. M., "Parallel Branch and Bound on MIMD System," Erasmus University, 1986, Rotterdam, The Netherlands.

[25] Van Roy, T. J., "Cross Decomposition for Mixed Integer Programming," *Math. Prog.*, Vol. 25(1983), pp. 46-63.

[26] Zenios, S. A., "Parallel Numerical Optimization : Current Status and an Annotated Bibliography," *ORSA Journal on Computing*. Vol. 1(1989), pp. 20-43.

[27] Zenios, S. A. and J. M. Mulvey,

"Nonlinear Network Programming on Vector Super Computers : A Study on the CRAY X-MP," *Operations Research*, Vol. 34(1986), pp. 667-682.

[28] Zenios, S. A., and J. M. Mulvey, "Vectorization and Multitasking of Nonlinear Network Programming Algorithms," *Mathematical Programming*, Vol. 42(1988), pp. 449-470.

[29] Zenios, S. A. and J. M. Mulvey, "A Distributed Algorithm for Convex Network Optimization Problems," *Parallel Computing*, Vol. 6(1988), pp. 45-56.