基于细粒度任务分配的空时自适应并行处理算法研究

王 超 刘 伟* 袁培苑
(北京理工大学信息与电子学院 北京 100081)

摘 要：对于空时自适应信号处理(Space-Time Adaptive Processing, STAP)算法的并行处理问题，传统方法以粗粒度的划分方式将 STAP 算法分配到特定硬件系统中的不同处理器中，利用处理器间的流水计算来提高系统计算吞吐量。该文分析了传统并行处理方法的缺陷：粗粒度的任务划分方式牺牲了 STAP 算法的并行度；传统处理方法仅能适用于特定的系统环境。针对上述情况，该文提出一种基于细粒度任务分配的 STAP 并行处理方法，该方法分为以下 3 个步骤：构建细粒度的 DAG(Direct Acyclic Graph)形式的 STAP 算法任务模型；使用统一拓扑结构模型描述不同结构的目标硬件系统；基于细粒度任务分配算法将任务模型分配到拓扑结构模型中的处理器实现并行计算。实验结果表明该并行处理方法能够达到良好的加速比，并且对于不同的 STAP 应用系统具有很好的适应性。

关键词：信号处理；空时自适应系统；并行处理；任务分配；细粒度
中图分类号：TN911.7 **文献标识码**： A **文章编号**：1009-5896(2012)06-1398-06
DOI: 10.3724/SP.J.1146.2011.00683

Research on the Parallel Processing Algorithm of STAP Based on Fine-grained Task Scheduling

Wang Chao Liu Wei Yuan Pei-yuan
(*School of Information and Electronics*, *Beijing Institute of Technology*, *Beijing* 100081, *China*)

Abstract: In the parallelization of Space-Time Adaptive Processing (STAP) arithmetic, traditional methods schedule the STAP arithmetic to different processors in the specific hardware architecture through coral-granularity division and improve the throughput by pipeline processing between processors. In the paper, its disadvantages are discussed from two perspectives: Coarse-grained scheduling hinders the parallelism; They are only suitable for the specific system parameters and hardware architectures. Thus, a new method based on fine-grained scheduling is put forward, which consists of three steps: Firstly, fine-grained task model in the form of Direct Acyclic Graph (DAG) is constructed; Secondly, the topology model is built to describe the target system; Finally, the established task model in fine-grained manner is assigned to different processors described in model topology. The experiment of the proposed method shows that it achieves better acceleration ratio, and more flexiable adaptation to different STAP applications.

Key words: Signal processing; Space-Time Adaptive Processing (STAP) systems; Parallel processing; Task scheduling; Fine-granularity

1 引言

空时自适应处理(STAP)是新一代相控阵雷达充分利用空域和时域信息通过空时2维滤波来抑制杂波与目标检测的一项关键技术，广泛应用于机载预警雷达、机载合成孔径雷达、机载战场侦察雷达及星载雷达、舰载雷达等实现杂波抑制与运动补偿[1]。STAP面对着根据外界杂波及干扰的环境实时地求解自适应权值向量的问题[2,3]。求解权值向量是一个高密集计算型问题，运算量巨大，数据交互复杂，通常采用并行处理技术提高算法的实时性。

传统的并行处理方法将 STAP 算法流程划分为若干粗粒度的计算任务，分配计算任务到专用的硬件环境中的处理器，利用处理器间的流水计算来提高系统计算吞吐量[4–6]。文献[7]提出了一种通用的 STAP 并行计算模型，并给出了适用于流水计算的粗粒度任务分配方法。粗粒度任务划分易于实现，但是牺牲了算法的并行度[8]。国外对于细粒度 STAP 并行处理也有一些研究。文献[9]引入遗传算法解决 STAP 并行化处理过程中的细粒度任务分配问题，但是该方法仅能适用于具有全交换拓扑结构的硬件环境中。文献[10]提出了一种细粒度数据划分方法，并在 GPU(Graphics Processing Unit)中实现了单指

2011-07-06 收到，2012-03-05 改回
*通信作者：刘伟 eliuwei@bit.edu.cn

令多数据流形式的 STAP 并行处理，但是 GPU 并不能应用于机载平台等嵌入式环境，这就限制了该方法实际应用性。

其次，传统的处理方法往往只适用于特定的 STAP 应用系统[4−6, 9]。特定应用系统中限定了系统参数，包括 STAP 处理通道数、脉冲数及处理距离单元数等信息；还限定了硬件环境信息，包括处理器的数量与处理器间的互联方式。当系统参数变化或硬件环境改变时，传统方法往往不再适用，这大大限制了传统方法的通用性。因此研究一种具有高并行度并且具有拓扑结构独立性的并行处理方法对于 STAP 算法的应用是非常有意义的。

我们可以分以下 3 个步骤实现 STAP 的并行处理：(1)划分 STAP 算法为细粒度的计算任务，建立细粒度的任务模型。相比于粗粒度的方式，细粒度的划分后任务模型具有更高的并行度[10]。(2)建立统一的拓扑结构模型，在模型参数中定义处理器的数量以及互联方式等信息，对于不同的目标硬件环境，配置不同的模型参数。(3)基于任务分配方法，完成任务模型到拓扑结构模型的分配。细粒度划分后计算任务的数量以及任务间的计算约束关系会变得非常复杂[10]。直接实现任务分配是非常困难的，更好的方法是选取一种合适的任务分配算法实现计算任务集到多处理器平台的映射。任务分配算法以优化加速比为目标通过计算任务间的约束关系及并行性选择将可并行的任务分配到不同的处理器实现独立计算。

2　基于细粒度任务分配的 STAP 处理方法

2.1 任务模型

分割 STAP 算法为细粒度的计算任务集，分割后各计算任务的计算粒度、任务间的数据传输关系以及任务间的执行逻辑顺序关系就可完全确定下来。一般使用 DAG 描述具有确定性的任务模型[11]。DAG=$\{V,E,\tau,c\}$，其中 V 为节点的集合，每一个节点都表示任务集中的一个计算任务；E 为边界的集合，节点间的边界表示任务间的数据传输；节点的权值 τ 表示节点的计算开销；边界的权值 c 表示数据传输的通信开销。

为了便于说明定义 $\mathrm{st}(v_i)$ 表示 v_i 的起始执行时间；$\mathrm{end}(v_i)$ 表示 v_i 结束执行时间；$\tau(v_i)$ 表示 v_i 的计算开销；e_{ij} 表示节点 v_i 到 v_j 的数据传输；$\mathrm{pr}(v_i)$ 表示 v_i 的所有前向节点集合；$\mathrm{cd}(v_i)$ 表示 v_i 的所有后向节点集合；$P(v_i)$ 表示节点 v_i 所分配到的处理器。那么如果 $e_{ij}\in E$，可以得到 $v_i\in\mathrm{pr}(v_j)$，$v_j\in\mathrm{cd}(v_i)$，并且

$$\mathrm{end}(v_i)=\mathrm{st}(v_i)+\tau(v_i) \tag{1}$$

在分配过程中，如果 $P(v_i)=P(v_j)$，且 $e_{ij}\in E$

$$\mathrm{st}(v_j)\geq\mathrm{end}(v_i) \tag{2}$$

如果 $P(v_i)\neq P(v_j)$，且 $e_{ij}\in E$

$$\mathrm{st}(v_j)\geq\mathrm{end}(v_i)+c(e_{ij}) \tag{3}$$

其中式(1)，式(2)与(3)保证分配结果满足 DAG 图中任务间执行约束关系。

2.2 拓扑结构模型

拓扑结构模型为目标硬件系统的抽象，包含了处理器的信息及处理器间的互联信息。定义拓扑结构模型 TP = $\{U,\mathrm{CH}\}$。其中 U 为有限的处理器集合，U 中的每个元素表示一个独立的处理器；CH 为有限的通信通道集合，CH 中的每个元素都表示处理器间的通信传输通道，传输通道分为单向传输通道和双向传输通道。对于不同的硬件环境，需要配置不同的 U 与 CH。

假设拓扑结构模型具有以下两个特性：(1)非抢占式：处理器只有在完成当前计算任务才能开始执行新的任务；(2)并发性。处理器可以同时执行并发的计算任务和通信传输任务。任务模型中的节点 V 需要被分配到拓扑结构模型中不同处理器 U。当 $u_k=P(v_i)$，$u_l=P(v_j)$，$e_{ij}\in E$ 且 $u_k\neq u_l$ 时 v_i 与 v_j 间的数据传输 e_{ij} 就转变为处理器间的 IPC(Inter Processor Communication)操作，并需要将该 IPC 操作分配到连接 u_k 与 u_l 的传输通道中。处理器间传输通道的选择一般采用最短路径准则。

2.3 任务分配算法

任务分配算法以优化加速比为目标将任务模型中的节点 v 分配到拓扑结构模型的处理器 u 中，并根据节点的分配完成 IPC 操作到传输通道的分配。任务分配是一个具有 NP(Non-deterministic Polynomial) 完备性的问题，很难获得最优的分配结果，因此常见的任务分配算法一般基于启发式算法获得次优结果[11]。

在现有的分配算法中，大都在分配过程中假定了理想的硬件环境[11−14]，即不限定处理器数目及互联通道数目，这并不符合于我们建立的拓扑结构模型。DLS(Dynamic Level Scheduling)是唯一具有拓扑结构独立性的分配算法[15]，即 DLS 脱离了拓扑结构对分配算法的影响。在分配过程中，DLS 实时计算分配不同节点到不同处理器的动态优先级 DL (Dynamic Level)，并依照 DL 顺序完成节点分配。因此我们选择 DLS 完成细粒度 STAP 任务模型的分配。

在每个分配步骤中，DLS 中以 Σ 表示当前状态

的已分配信息，分配节点 v_i 到处理器 u_j 时需要完成：(1)根据 $\mathrm{pr}(v_i)$ 的分配信息，完成 v_i 前向 IPC 操作分配；(2)完成 v_i 到 u_j 的分配。Σ 包含已分配节点以及 IPC 的 st，end 等分配信息。统计出 v_i 执行前所需要完成的传输任务集合，如下：

$$\mathrm{recv\_IPC}(v_i)=\{e_{ki}\mid v_k\in \mathrm{pr}(v_i),P(v_k)\neq u_j\} \quad (4)$$

IPC 操作只能够分配在拓扑结构模型中定义的通信通道，并根据各通信通道的已分配状态确定 IPC 操作的起始执行时间。DA 表示所有 $\mathrm{recv\_IPC}(v_i)$ 的最后结束时间，同时也表示了 Σ 下 v_i 在 u_j 上的数据就绪时间。

$$\mathrm{DA}(v_i,u_j,\Sigma)=\max\{\mathrm{end}[\mathrm{recv\_IPC}(v_i)]\} \quad (5)$$

由 DA 可以计算出 $\mathrm{st}(v_i)$，代入式(3)计算式 $\mathrm{end}(v_i)$ 完成 v_i 在 p_j 上的分配。当前阶段结束后更新 Σ。Σ 的迭代更新确保每个分配阶段都可以完全获取各处理器和通信通道的任务分配信息，并依此完成新任务的分配。式(6)中 $\mathrm{TF}(u_j,\Sigma)$ 表示 Σ 下 u_j 的空闲时间。

$$\mathrm{st}(v_i)=\max[\mathrm{DA}(v_i,u_j,\Sigma),\mathrm{TF}(u_j,\Sigma)] \quad (6)$$

分配过程中 DLS 使用 Σ 管理当前状态的已分配任务信息，并约束新分配的计算任务与通信任务不能和已分配的任务产生冲突。这就避免了处理器中计算任务间的竞争与通信通道中 IPC 任务间的竞争。分配完成后，所有节点到处理器的映射信息，IPC 操作到通信通道的映射信息，计算节点与 IPC 的 st 及 end 都可以完全确立下来。此时可得到各处理节点 $\mathrm{load}(u_i)$ 的计算负载与通信通道的通信负载 $\mathrm{load}(ch_i)$：$\mathrm{load}(u_i)=\mathrm{st}(u_i)-\mathrm{end}(u_i)$，$\mathrm{load}(ch_i)=\mathrm{st}(ch_i)-\mathrm{end}(ch_i)$。其中 $\mathrm{st}(u_i)$ 表示处理器 u_i 上第 1 个计算节点的起始时间；$\mathrm{end}(u_i)$ 表示 u_i 上最后一个计算节点的结束时间；$\mathrm{st}(ch_i)$ 表示通信通道 ch_i 第 1 个 IPC 任务的起始时间；$\mathrm{end}(ch_i)$ 表示 ch_i 上最后一个 IPC 的结束时间。分配结束后通过计算负载与通信负载计算加速比 ACR 评价分配性能，如式(7)所示：

$$\mathrm{ACR}=\sum_{v_i\in V}\tau(v_i)\Big/\max\left\{\max_{u_i\in U}[\mathrm{load}(u_i)],\max_{ch_i\in \mathrm{CH}}[\mathrm{load}(ch_i)]\right\} \quad (7)$$

综上所述，基于细粒度任务分配的 STAP 并行处理方法分为 3 个步骤：(1)构建任务模型。根据系统参数建立 STAP 处理流程图，将其划分为细粒度的任务集并建立任务 DAG 图。DAG 中每个节点的计算粒度确定后，其计算开销以及边界的传输开销可以通过实际测试获取。(2)构建拓扑结构模型。拓扑模型 $\mathrm{TP}=\{U,\mathrm{CH}\}$ 参数中应明确定义处理器的个数以及处理器间的互联关系。根据目标硬件环境配置拓扑结构模型参数。(3)任务分配过程。使用 DLS 算法实现任务模型到拓扑结构模型的映射。可由式(7)估算分配结果的加速比。最终输出并行分配结果；并行分配结果中包括：计算任务到处理器的映射关系、IPC 任务到传输通道的映射关系以及计算任务和 IPC 的执行顺序关系。

3 并行处理的实现

基于细粒度任务分配的并行处理方法分 3 步完成，其中任务分配过程使用较为成熟的 DLS 分配算法，因此实现 STAP 应用系统的并行处理时需要完成细粒度任务模型的构建以及拓扑结构模型的配置。

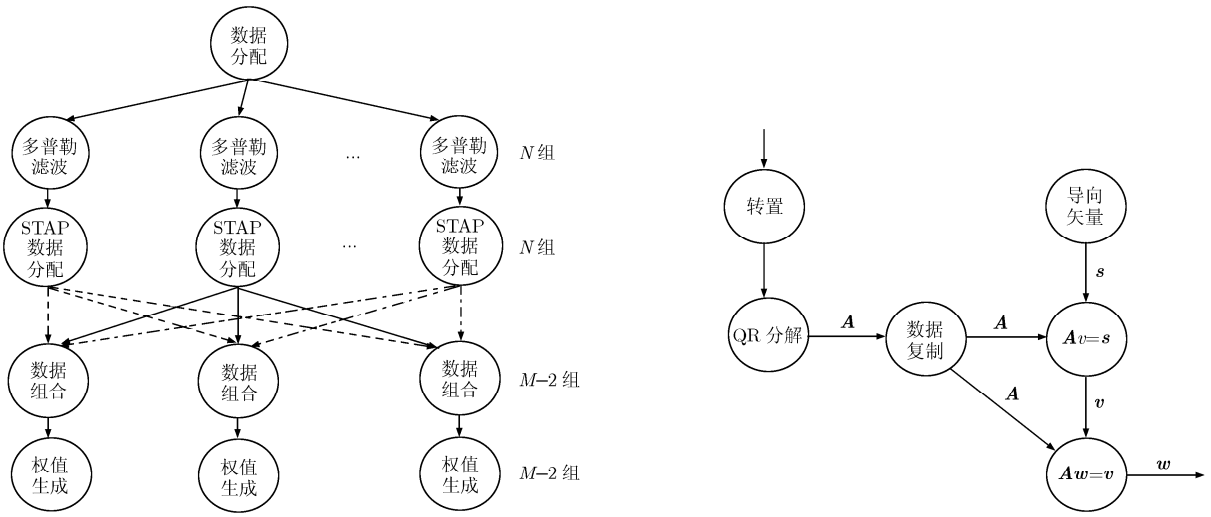
3.1 细粒度任务模型构建

全自适应 STAP 算法计算量过于庞大，工程应用一般选择基于频域降维的 3DT-SAP 算法。本节以 3DT-SAP 算法为例简述细粒度 STAP 任务模型的构建方法。设置脉冲数为 M，阵元数为 N，距离单元数为 L，参照文献[4,5]中的 STAP 处理流程构建粗粒度的任务 DAG 图，如图 1(a)所示。图中的节点表示计算任务，节点间的连接箭头表示计算任务间的数据传输。

由数据分配节点将数据分发到 N 个多普勒滤波节点，每个节点完成 L 次 M 点 FFT，完成后数据由 STAP 数据分配节点分配到 $M-2$ 个数据组合节点中。对于每个数据组合节点，将 N 个处理通道中相邻的 3 组脉冲数据组合在一起[15]。组合后的数据由权值生成节点计算得到自适应权值。权值生成可以通过如下步骤完成：组合数据转置后经由 QR 分解节点得到 $3N\times 3N$ 维的上三角矩阵 $\boldsymbol{A}$；联合空时 2 维导向矢量 $\boldsymbol{s}$ 求解两次线性方程组计算得到自适应权值矢量 $\boldsymbol{w}$，如图 1(b)所示。最终由自适应权值联合数据组合节点的输出完成滤波过程，得到 STAP 的输出。

传统的 STAP 并行处理方法通常使用粗粒度 QR 分解操作，牺牲了算法的并行度，本文将采用一种细粒度的 QR 分解方法。计算权值的距离门数 L_{ls} 应满足自适应权值的收敛条件，取 $L_{ls}=3\times 3N$ 构成 $9N\times 3N$ 的数据矩阵来求解权值向量[4]。将输入 $L_{ls}\times 3N$ 阶矩阵依照行顺序分为 K 块 $(L_{ls}/K)\times 3N$ 阶子矩阵，在实际应用中一般保证 $3N$ 为 L_{ls}/K 的整数倍；将 QR 分解分割为两种细粒度的计算任务：消除下三角元素操作以及消除上三角元素操作。

(1)消除下三角操作过程：假设子矩阵 $\boldsymbol{M}1$ 的前



(a) 任务 DAG 图　　(b) 权值生成流程示意图

图 1 粗粒度任务模型 DAG

r 列元素全部为 0，$r+1$ 列元素非 0，使用 Givens 旋转法将元素 $\boldsymbol{M}1_{ij}$ ($i\in[2,L_{ls}/K]$，　$j\in[r+1,\ r+i-1]$)消除为 0。

(2)消除上三角操作过程：选择全 0 列数 r 相同的两个子矩阵 $\boldsymbol{M}1$ 和 $\boldsymbol{M}2$，矩阵中元素 $\boldsymbol{M}1_{ij}$ 和 $\boldsymbol{M}2_{ij}$ ($i\in[2,L_{ls}/K]$，　$j\in[r+1,r+i-1]$)为 0；元素 $\boldsymbol{M}1_{ij}$ 和 $\boldsymbol{M}2_{ij}$ ($i\in[1,L_{ls}/K]$，　$j\in[r+1,r+i]$)非 0。以 $\boldsymbol{M}1$ 作为参考矩阵将矩阵 $\boldsymbol{M}2$ 中的元素 $\boldsymbol{M}2_{ij}$ ($i\in[1,L_{ls}/K]$，　$j\in[r+1,r+i]$)消除为 0。完成后 $\boldsymbol{M}1$ 的全 0 列数仍为 r，$\boldsymbol{M}2$ 的全 0 列数为 $r+L/K$。

对于第 k 个子矩阵 $\boldsymbol{M}k$，如果 $k\le 3N/(L/K)$ 时，分解完成的条件为：前 $(i-1)\times(L/K)$ 列元素全为 0，且 $\boldsymbol{M}k_{ij}$ ($i\in[2,L_{ls}/K]$，　$j\in[r+1,r+i-1]$)为 0，元素 $\boldsymbol{M}k_{ij}$ ($i\in[1,L_{ls}/K]$，　$j\in[r+1,r+i]$)非 0；$k>L_{ls}/K\times 3N$ 时，完成条件为 $\boldsymbol{M}k$ 消除为全 0 矩阵。

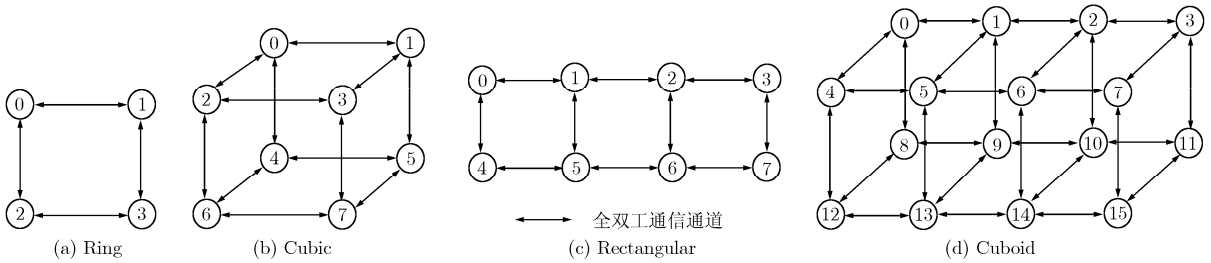
对于所有子矩阵 $\boldsymbol{M}k$ ($k\in[1,K]$)，持续进行消除下三角操作和消除上三角操作，直到满足分解完成条件。当所有 K 个子矩阵都完成分解后，依照行序号重新将子矩阵组合即可得到新的矩阵，其前 $3N$ 行 $3N$ 列为上三角矩阵，其余部分为 0。将图 1(b)中的 QR 分解节点替换为上述分解方法，以此构建细粒度的 DAG 图，完成任务模型建立。

3.2 拓扑结构模型的配置

定义 4 种测试拓扑结构：Ring, Cubic, Rectangular 及 Cuboid 如图 2 所示。图中每个节点表示一个处理器，处理器间的箭头线表示两个处理器间全双工的通信通道。Ring 型拓扑结构硬件上由 4 片 TS201 DSP 组成，每片 DSP 与相邻的 2 片 DSP 通过 LINK 通道双向互联；Cubic 型拓扑结构硬件上由 8 片 TS201 DSP 组成，每片 DSP 与相邻的 3 片 DSP 通过 LINK 通道双向互联，Cubic 型可以看作是 Ring 型结构的扩展。Rectangular 型拓扑结构硬件上 8 片 TS201 DSP 组成，相邻的 DSP 之间通过 LINK 通道双向互联；Cuboid 型硬件上由两组 Rectangular 型拓扑结构扩展而成。依据图 2 各拓扑结构中处理器的个数以及处理器间的连接关系配置拓扑结构模型 $\mathrm{TP}=\{U,\mathrm{CH}\}$。

3.3 实验及分析

依照表 1 配置 5 组 STAP 处理参数，并使用 DLS 算法实现并行任务分配。其中实验 1–实验 4 依据 4.1 节中的方法建立细粒度的 DAG 任务图，实验 5 直



(a) Ring　(b) Cubic　(c) Rectangular　(d) Cuboid

图 2 拓扑结构示意图

表 1 STAP 参数设定及 p 与 CCR 统计

实验	N	M	L	K	子矩阵维数	p	CCR
1	8	32	256	3	24×24	36.82	0.21
2	8	16	256	3	24×24	20.41	0.21
3	16	16	256	6	24×48	35.27	0.13
4	16	16	512	6	24×48	34.04	0.20
5	16	16	256	1	144×48	14.36	0.10

接使用粗粒度的 DAG 任务图。

通过实际测试获取各 DAG 图中各任务在 TS201 中的实际计算开销，表 2 统计了 TS201 中各维数矩阵消除上三角与消除下三角操作的计算开销。其中实验 5 中 QR 分解分块个数 $K=1$，因此仅需要一次消除下三角操作即可完成 QR 分解，故不存在消除上三角操作。根据式(1)与式(2)统计得到 5 个实验中 DAG 的并行度 p 及 CCR，统计如表 1 所示。

表 2 QR 分解操作计算开销统计 (μs)

子矩阵维数	消除上三角操作	消除下三角操作
24×48	680	620
24×24	549	505
144×48	–	15431

由图 1(a)可以看出，$M-2$ 组权值计算之间为相对独立的过程。在实验 1 中 $M=32$，需完成 30 组权值计算与 STAP 滤波，因此实验 1 中的并行度最高达到了 36.82。对比实验 3 与实验 5，实验 3 中采用细粒度的任务划分方法，并行度达到了 35.27；而实验 5 采用粗粒度的方法，并行度仅为 14.36。可以看出细粒度的任务划分虽然增加了传输开销，但是换取了并行度的有效提高。

采用细粒度任务模型后，STAP 算法中各处理阶段被拆分为细粒度的计算任务与通信任务，并映射到不同的处理器交错执行，很难直接评估各处理阶段的计算开销时间与通信开销。因此一般使用加速比 ACR 来衡量各实验中 STAP 的总体并行性能，ACR 由式(9)计算得出。图 3 给出了 5 组实验中 DLS 的 ACR 统计，其中每组实验都包含了 Ring, Cubic, Rectangular 以及 Cuboid 4 种不同拓扑结构下的分配测试。

可以看出：(1)在同一组实验中，随着拓扑结构中处理器节点个数的增加，ACR 也逐步增加。(2)对比实验 1 与实验 3。虽然实验 1 中任务图的 $p=36.82$ 大于实验 3 的 $p=35.27$，但是实验 1 的

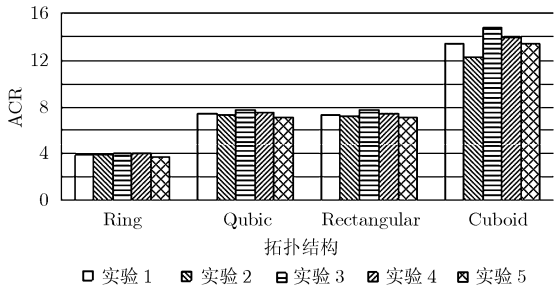


图 3 ACR 统计

CCR 较大，因此在 4 种拓扑结构下实验 3 的 ACR 都超越了实验 1。可以看出在 STAP 算法并行度接近的情况下，较大的通信开销将会影响并行加速性能。(3)对比实验 3 与实验 5，在相同的系统参数配置下，细粒度任务模型具有更高的并行性，更适合于并行实现，因此达到了更优的 ACR。

参考文献[7]提出了一种适用于流水处理的通用 STAP 并行计算模型，并给出了该计算模型下粗粒度的计算任务分配方法。该方法下的并行加速比统计见表 3。由于并行加速比与处理器的数量有直接关系，因此我们选择具有相同处理器数量的拓扑结构模型实验结果与参考文献[7]的加速比进行比较。其中 4 片处理器对应于本文中 Ring 型拓扑结构下的 5 组实验；8 片处理器对应于本文中 Cubic 与 Rectangular 型拓扑结构下的 5 组实验。结果比较如表 3 所示。

表 3 加速比比较

拓扑结构	Ring	Cubic	Rectangular
实验1	3.87	7.43	7.45
实验2	3.87	7.21	7.24
实验3	3.95	7.82	7.76
实验4	3.90	7.59	7.48
实验5	3.66	7.16	7.12
文献[7]实验	3.67	7.31	

由于细粒度 STAP 算法本身具有很高的并行度，在处理器数量都为 4 的条件下，本文中 Ring 型拓扑结构模型的前 4 组实验使用了细粒度的任务模型，加速比性能要优于参考文献[7]中的加速比 3.67；第 5 组实验使用了粗粒度的任务模型，因此加速比与参考文献[7]基本一致。

在 Cubic 与 Rectangular 下的实验中，实验 1，实验 3 和实验 4 的加速比优于参考文献[7]中 8 片处理器的实验结果，实验 2 的加速性能基本与其一致。而第 5 组实验使用了粗粒度的任务模型，因此加速

比小于参考文献[7]中的结果。

联合表 1 可以看出，5 组实验中实验 2 与实验 5 的任务模型并行度较低，因此加速比性能较差。而实验 1，实验 3 和实验 4 的任务模型并行度较高，因此在与参考文献[7]的结果比较中达到了更优的加速性能。这也再次说明，并行度越高的任务模型越适合于本文提出的 STAP 并行处理方法。

5 组实验加载了不同的系统参数，并且每组实验都使用了 4 种完全不同的拓扑结构。由结果可以看出：(1)建立细粒度的任务模型提高了算法的并行度，DAG 形式的任务模型适应于不同系统参数的 STAP 应用；(2)构建 TP = {U,CH} 形式的拓扑结构可以适用于不同目标硬件环境；(3)选择具有拓扑结构独立性的 DLS 分配算法使得整个 STAP 并行处理过程脱离了应用参数与系统硬件结构的限制。

4　结论

在 STAP 并行处理算法中，传统的并行处理方法使用粗粒度的任务划分方式，并且仅仅能够适用于特定的应用参数及硬件系统结构。粗粒度的任务划分牺牲了 STAP 流程的并行度；限定应用参数及硬件结构虽然易于实现，但同时也限定了传统的 STAP 并行处理方法的通用性。针对这些问题，本文提出了一种更具实用性的 STAP 并行处理方法，该方案分为以下 3 个步骤：建立 STAP 处理流程并以此构建细粒度的 DAG 形式任务模型；根据实际硬件构建拓扑结构模型；基于 DLS 任务分配算法将任务模型中的任务分配到拓扑结构模型中的不同处理器实现 STAP 并行计算。

由实验结果可以看出，该并行方法能够达到良好的加速比，并且对于不同的 STAP 应用以及不同的硬件环境具有很好的适应性。

参 考 文 献

[1] 保铮, 廖桂生, 吴仁彪, 等. 相控阵机载雷达杂波抑制的时空二维自适应滤波[J]. 电子学报, 1993, 21(9): 1–7.
Bao Zheng, Liao Gui-sheng, Wu Ren-biao, *et al.*. 2-D temporal-tpatial adaptive clutter suppression for phased array airborne radars[J]. *Acta Electronica Sinica*, 1993, 21(9): 1–7.

[2] Huang Yao. A reduced-rank STAP method based on solution of linear equations[C]. Proceedings of the 2010 International Conference on Computer Design and Applications (ICCDA), Qinghuangdao, China, 2010: 235–238.

[3] Wu Ren-biao, Wang Lu, and Su Zhi-gang. Study on adaptive monopulse with reduced dimension STAP technique[C]. Proceedings of the 2010 International Conference on Image Analysis and Signal Processing (IASP), Xiamen, China, 2010: 159–163.

[4] 范西昆, 王永良, 陈辉. 机载雷达空时自适应处理的实时实现[J]. 电子与信息学报, 2006, 28(12): 2224–2227.
Fan Xi-kun, Wang Yong-liang, and Chen Hui. Real-time implementation of airborne radar space-time adaptive processing[J]. *Journal of Electronics & Information Technology*, 2006, 28(12): 2224–2227.

[5] 任磊, 王永良, 陈辉, 等. STAP 并行处理系统的调度问题研究[J]. 系统工程与电子技术, 2009, 31(4): 874–880.
Ren Lei, Wang Yong-liang, Chen Hui, *et al.*. Research on the scheduling problems of STAP parallel processing system[J]. *Systems Engineering and Electronics*, 2009, 31(4): 874–880.

[6] Lebak J M and Bojanczyk A W. Design and performance evaluation of a portable parallel library for space-time adaptive processing[J]. *IEEE Transactions on Parallel and Distributed Systems*, 2000, 11(3): 287–298.

[7] 邵银波, 王永良, 李强, 等. 一种用于空时自适应处理的并行计算模型[J]. 电子学报, 2006, 34(3): 450–453.
Shao Yin-bo, Wang Yong-liang, Li Qiang, *et al.*. A parallel computation model for space-time adaptive processing[J]. *Acta Electronica Sinica*, 2006, 34(3): 450–453.

[8] West M and Antonio K. A genetic algorithm approach to scheduling communications for a class of parallel space-time adaptive processing algorithms[J]. *Journal of Parallel and Distributed Computing*, 2002, 62(9): 1386–1406.

[9] Roedera M, Davisa N, Furteka J, *et al.*. GPU implementations for fast factorizations of STAP covariance matrices[C]. Proc. SPIE, San Diego, USA, 2008: 707403-1–707403-12.

[10] Kruatrachue B and Lewis T. Grain size determination for parallel processing[J]. *IEEE Transactions on Software*, 1988, 5(1): 23–32.

[11] Wang Chao and Liu Wei. Multi-processor task scheduling in signal processing systems[C]. Proceedings of the International Conference on Computer Science and Information Technology, Chengdu, China, 2011: 532–539.

[12] Ebaid A, Ammar R, and Rajasekaran S. Task clustering & scheduling with duplication using recursive critical path approach (RCPA)[C]. Proceedings of the 2010 IEEE International Symposium on Signal Processing and Information Technology, Luxor, 2010: 34–41.

[13] Hwang R, Gen M, and Katayama H. A comparison of multiprocessors task scheduling algorithms with communication costs[J]. *Computer & Research*, 2008, 35(3): 976–993.

[14] Sun Wei-fang, Zhu Yu-dan, Sun Zhi-yuan, *et al.*. A priority-Based task scheduling algorithm in Grid [C]. Proceedings of the Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), Dalian, China, 2010: 311–315.

[15] Sih G C and Lee E A. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architecture[J]. *IEEE Transactions on Parallel and Distributed Systems*, 1993, 4(2): 175–186.

王　超：　男，1985 年生，博士生，研究方向为实时信号处理.
刘　伟：　男，1976 年生，讲师，研究方向为实时信号处理.
袁培苑：　女，1988 年生，硕士生，研究方向为实时信号处理.