

آشکارسازی سیگنال بر اساس پردازش موازی

مبتنی بر جی‌پی‌یو در شبکه‌های حس‌گری صوتی

دارای زیرساخت

حامد صادقی^{۱*} و امیر اخوان^۲

^۱ آزمایشگاه مخابرات بی‌سیم، دانشکده برق و کامپیوتر، دانشگاه تربیت مدرس، تهران، ایران

^۲ گروه مهندسی پزشکی، دانشگاه صنعتی همدان، همدان، ایران

چکیده

الگوریتم فیشر، یکی از معروف‌ترین و پرکاربردترین روش‌های آشکارسازی آرایه‌ای سیگنال‌های صوتی بسامد پایین در شبکه‌های حس‌گری دارای زیرساخت است؛ اما یکی از مشکلات عمده در به‌کارگیری این الگوریتم، زمان طولانی انجام پردازش در آن است که در عمل، پیاده‌سازی بلادرنگ آشکارساز را با مشکل مواجه می‌سازد. در این مقاله، چگونگی پیاده‌سازی الگوریتم فیشر را با استفاده از واحد پردازش گرافیک (جی‌پی‌یو) به‌منظور تحقق محاسبات سریع و انجام پردازش‌های نزدیک به زمان واقعی، ارائه می‌کنیم. به‌خصوص به‌منظور بهبود هرچه بیشتر سرعت محاسبات، الگوریتم آشکارسازی با استفاده از روش پردازش موازی (مبتنی بر جی‌پی‌یو) پیاده‌سازی شده است. نتایج شبیه‌سازی‌ها، ارتقای قابل ملاحظه سرعت آشکارساز فیشر را نشان می‌دهند که باعث بهبود کارایی شبکه حس‌گری صوتی خواهد شد.

واژگان کلیدی: شبکه حس‌گری، پردازش آرایه‌ای، شکل‌دهی پرتو، پردازش موازی، جی‌پی‌یو.

Signal Detection Based on GPU-Assisted Parallel Processing for Infrastructure-based Acoustical Sensor Networks

Hamed Sadeghi^{1*} & Amir Akhavan²

¹ Wireless Communication Laboratory, ECE Department, Tarbiat Modares University, Tehran, Iran

² Department of Biomedical Engineering, Hamedan University of Technology, Hamedan, Iran

Abstract

Nowadays, several infrastructure-based low-frequency acoustical sensor networks are employed in different applications to monitor the activity of diverse natural and man-made phenomena, such as avalanches, earthquakes, volcanic eruptions, severe storms, super-sonic aircraft flights, etc. Two signal detection methods are usually implemented in these networks for the purpose of event occurrence identification, which are the progressive multi-channel correlator (PMCC) and the so-called Fisher detector. But, the Fisher method is more important and applicable in low signal-to-noise (SNR) ratio conditions, which is of a special interest in acoustical monitoring networks. Unfortunately, an important disadvantage of this algorithm is its relative high detection-time; which limits its application for real-time detection scenarios. This disadvantage is fundamentally due to a beam forming process in Fisher algorithm, which requires doing complete search in a slowness-network, constructed from possible incoming wave front directions and speeds. To address this issue, we propose a method for implementation of this beam forming on a graphics processing unit (GPU), in order

* Corresponding author

* نویسنده عهده‌دار مکاتبات

to realize a fast-computing and/or near real-time signal processing technique. In addition, we also propose a parallel-processing algorithm for further enhancement of the performance of this GPU-based Fisher detector. Simulation results confirm the performance improvement of Fisher detector, in terms of required processing time for acoustical signal detection applications.

Keywords: Sensor network, array processing, beamforming, parallel processing, GPU.

۱- مقدمه

یکی از انواع شبکه‌های حس‌گری^۱، شبکه‌های حس‌گری صوتی مبتنی بر زیرساخت^۲ (IASN) است که بیش‌تر به‌منظور آشکارسازی و جهت‌یابی سیگنال‌های گسیل‌شده از منبع مولد صدا به‌کار می‌روند [1]-[4]. در IASN پس از دریافت سیگنال‌های ارسالی از حس‌گرها در مرکز هم‌جوشی^۳ (FC)، وقوع رخداد با استفاده از الگوریتم‌های پردازش آرایه‌ای جستجو می‌شود. این الگوریتم‌ها شامل دو مرحله اصلی آشکارسازی^۴ و سپس تخمین جهت و سرعت ورود موج است. اصول پردازش سیگنال در این الگوریتم‌ها بر مبنای استفاده از مفهوم هم‌دوسی^۵ امواج ساطع‌شده از منبع صوت و نیز هم‌بستگی متقابل بین سیگنال‌های خروجی حس‌گرها است. به‌عبارت دیگر، از آن‌جایی‌که در صورت حضور سیگنال مورد نظر در موج‌دیس‌های ثبت‌شده، هم‌بستگی بین موج‌دیس‌های خروجی حس‌گرها افزایش می‌یابد، لذا پارامتر هم‌بستگی به‌عنوان یک معیار مناسب برای جداکردن نوفه از سیگنال هدف، می‌تواند استفاده شود. از طرفی به‌منظور تخمین اختلاف زمان ورود سیگنال به حس‌گرهای متفاوت از هم‌بستگی متقابل بین سیگنال‌های حس‌گرهای مختلف می‌توان استفاده کرد. روشن است که مقادیر اختلاف زمانی، به‌هندسه و فاصله نسبی حس‌گرها از یکدیگر و نیز پارامترهای موج صوتی وابسته است. پس از تخمین مقادیر اختلاف زمانی، پارامترهای انتشار موج از جمله راستای ورود موج و سرعت آن محاسبه می‌شود. یکی از روش‌های پرکاربرد آشکارسازی سیگنال و تخمین پارامترهای موج در IASN، الگوریتم فیشر^۶ است.

این روش به‌دلیل پایداری مناسب در مقابل نوفه و قابلیت پیاده‌سازی پهن‌بند به‌طور گسترده برای تعیین جهت و آشکارسازی سیگنال‌های صوتی و لرزه‌ای به‌کار می‌رود [5]-[8]. روش فیشر برای نخستین‌بار در سال ۱۹۵۷ توسط ملتن^۷ و بایلی^۸ پیشنهاد شد [9]. پس از آن این الگوریتم به‌منظور آشکارسازی سیگنال‌های لرزه‌ای استفاده شد [10]. این آشکارساز که بر اساس آنالیز واریانس نسبت سیگنال به نوفه^۹ استخراج می‌شود، شامل دو بخش شکل‌دهی پرتو^{۱۰} و آشکارسازی مبتنی بر نسبت فیشر (به‌منظور ارزیابی محتوای سیگنال پرتوها) است. نسبت فیشر درواقع نسبت واریانس سیگنال به واریانس نوفه است که در هر دو حوزه زمان و بسامد قابل پیاده‌سازی است. یکی از ویژگی‌های برجسته این روش آشکارسازی، مقاوم‌بودن آن در مقابل نوفه است. این خاصیت در پردازش داده‌های مربوط به زلزله و شرایطی که نسبت توان سیگنال به نوفه پایین است، اهمیت زیادی دارد. شایان ذکر است که آشکارساز فیشر در کاربردهای مختلفی هم‌چون تشخیص فعالیت‌های آتشفشانی و ردیابی مسیر شهاب‌های آسمانی نیز به‌کار می‌رود [11]-[12]. این روش در مقایسه با آشکارساز معروف و پرکاربرد هم‌بستگی چندکاناله پیشرو^{۱۰}، موسوم به پی‌ام‌سی‌سی [13]، توانایی بیش‌تری برای آشکارسازی سیگنال در نسبت سیگنال به نوفه‌های پایین دارد. در مقابل، الگوریتم فیشر به‌دلیل جستجوی صفحه‌گندی به زمان بیش‌تری برای پردازش داده نیاز دارد.

در روش آشکارساز فیشر تخمین پارامترهای موج دریافتی بر مبنای اختلاف زمانی بین ثبت‌های صورت‌گرفته در حس‌گرها انجام می‌شود. از آن‌جایی که این اختلاف زمانی، به چیدمان حس‌گرهای آرایه و جهت ورود سیگنال بستگی دارند، در این روش ثبت‌های حس‌گرها به‌زای یک

¹ Sensor Networks (SNs)

² Infrastructure-based Acoustic Sensor Network (IASN)

³ Fusion Center (FC)

⁴ Detection

⁵ Coherency

⁶ Fisher

⁷ Melton and Bailey

⁸ Signal-to-Noise Ratio (SNR)

⁹ Beamforming

¹⁰ Progressive Multi-Channel Correlation (PMCC)

بسیار حائز اهمیت است. به‌عنوان جمع‌بندی، این روش یک الگوریتم تلفیقی شکل‌دهی پرتوی سریع مبتنی بر آمارگان فشر خواهد بود که به‌طور همزمان، هم حضور سیگنال هدف را آشکارسازی کرده و هم آن را می‌تواند جهت‌یابی کند. ساختار مقاله بدین شرح است: در بخش ۲، مفاهیم اولیه پردازش موازی داده، ساختار پردازنده گرافیکی و شیوه برنامه‌نویسی آن شرح داده شده است. همچنین در این بخش مروری مختصر بر آشکارسازی سیگنال‌های بسامد-پایین با استفاده از نسبت فشر انجام می‌شود. در بخش ۳ روش پیشنهادی آشکارسازی سریع منابع صوتی بسامد-پایین و چگونگی پیاده‌سازی الگوریتم آن ارائه شده است. نتایج شبیه‌سازی‌ها و کارایی روش پیشنهادی در بخش ۴ بررسی می‌شوند. در انتها نتیجه‌گیری کلی مقاله و مراجع در بخش ۵ و ۶ ارائه شده است.

۲- آشکارسازی سریع سیگنال‌های صوتی

در همین‌اواخر استفاده از واحد پردازنده گرافیکی (جی‌پی‌یو)^۷ مورد توجه پژوهش‌گران حوزه محاسبات و پردازش سریع قرار گرفته است. مقالات منتشرشده در این زمینه حاکی از آن است که به‌کارگیری این پردازنده‌ها به‌منظور تسریع الگوریتم‌های دارای قابلیت موازی‌سازی، موفق بوده است [14]-[19]. به‌طور خاص، در حال حاضر یکی از زمینه‌های پژوهشی فعال در حوزه‌هایی که نیازمند پردازش سریع هستند و قابلیت موازی شدن در الگوریتم‌های مورد استفاده آن‌ها وجود دارد، به‌کارگیری پردازنده‌های گرافیکی با معماری کودا^۸ است [20]، [21]. در این مقاله پیشنهاد می‌شود، از این پردازنده‌ها به‌منظور پردازش سریع سیگنال‌های صوتی دریافتی از IASN^۷ها استفاده شود. به‌عبارت دیگر در این مقاله، الگوریتم فشر با استفاده از پردازش موازی مبتنی بر جی‌پی‌یو پیاده‌سازی شده است. نتایج مقاله نشان می‌دهد که با این رویکرد، نقطه ضعف آشکارساز فشر، که زمان‌بر بودن جستجو در شبکه گندی است، برطرف شده است.

پیش از پرداختن به الگوریتم پیشنهادی، ابتدا برخی مفاهیم پایه‌ای معرفی می‌شوند.

جهت و سرعت فرضی برای ورود سیگنال، هم‌تراز می‌شوند؛ سپس هم‌دوسی آن‌ها با استفاده از نسبت فشر محاسبه می‌شود. این کار به‌طور مداوم به‌ازای جهت‌ها و سرعت‌های مختلف ورود سیگنال صدا صورت می‌پذیرد تا درنهایت، جهت و سرعتی که بیشینه مقدار هم‌دوسی بین ثبت‌ها را ایجاد کند، به‌عنوان جهت و سرعت موج انتخاب شود. یک روش جستجوی سامان‌مند^۱ برای تخمین جهت و سرعت موج، استفاده از روش شکل‌دهی پرتو است که در آن از یک شبکه گندی^۲ استفاده می‌شود. اندازه هر بردار در صفحه گندی، معکوس سرعت ظاهری^۳ است و زاویه آن با شمال زاویه سمت‌پشتی^۴ نامیده می‌شود. جستجوی بردار گندی در تمامی جهت‌ها، تضمین می‌کند که یکی از پرتوها جهت تقریبی ورود سیگنال به آرایه را نشان دهد. واضح است که بسته به تعداد نمونه‌ها در صفحه گندی، میزان تفکیک‌پذیری^۵ زاویه و سرعت تخمینی موج در این شبکه تحت تاثیر قرار خواهد گرفت. از سوی دیگر با افزایش تعداد نمونه‌ها، زمان بیشتری نیز برای محاسبات مورد نیاز است و این مانع از پیاده‌سازی بلادرنگ آشکارسازی سیگنال‌های صوتی می‌شود.

از آنجایی که مشکل اصلی روش فشر، انجام جستجوی سنگین^۶ در فضای شبکه گندی و در نتیجه، غیر زمان واقعی بودن سامانه است، در این مقاله یک روش سریع‌سازی الگوریتم فشر پیشنهاد می‌شود که دارای کاربردهای گسترده‌ای در سامانه‌های نظارتی خواهد بود. روش کار بدین‌صورت است که ابتدا برخی محاسبات اولیه و نیز پنجره‌گذاری بر روی داده‌های ورودی در سی‌پی‌یو انجام شده و سپس برای هر پنجره، نسبت‌های آمارگان فشر به‌ازای هر کدام از عناصر شبکه گندی، به‌طور موازی توسط جی‌پی‌یو محاسبه شوند. درنهایت، بردار گندی متناظر با بیشینه نسبت فشر، انتخاب شده و با استفاده از آن پارامترهای موج برای آن پنجره زمانی تخمین زده می‌شوند. در این مقاله نشان داده خواهد شد که روش پیشنهادی باعث کاهش زمان آشکارسازی منبع شد، که این امر برای سامانه‌های نظارتی

¹ Systematic

² Slowness

³ Apparent velocity

⁴ Back azimuth (BAZ)

⁵ Resolution

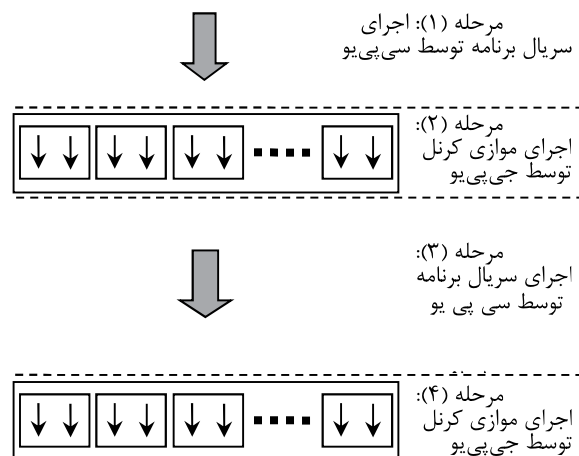
⁶ Exhaustive search

⁷ Graphics Processing Unit (GPU)

⁸ Compute Unified Device Architecture (CUDA)

۲-۱- موازی سازی داده ها

کاربردها، ابداع شد. با استفاده از این معماری، جی پی یو می تواند علاوه بر محاسبات گرافیکی، محاسبات همه منظوره را نیز انجام دهد. شرکت انویدیا علاوه بر ارائه سخت افزاری که بتواند محاسبات گرافیکی و همه منظوره را انجام دهد، زبان برنامه نویسی جدیدی به نام کودا-سی^۵ را نیز ارائه کرد. این نوع برنامه نویسی که در قالب محاسباتی غیرهمگن^۶ انجام می شود، امکان بهره گیری همزمان از مزایای سی پی یو و جی پی یو را فراهم می سازد. در این روش برنامه نویسی، بخش سری کد توسط سی پی یو اجرا شده و هسته موازی برنامه، موسوم به کرنل، توسط جی پی یو پردازش می شود. مدل اجرایی این روش برنامه نویسی در (شکل-۱) نمایش داده شده است. با توجه به این شکل، سی پی یو کدهای مربوط به مراحل ۱ و ۳ را به صورت سریال و جی پی یو کدهای مربوط به مراحل ۲ و ۴ را به صورت موازی اجرا می کند. هنگامی که یک تابع کرنل به صورت موازی راه اندازی^۷ شود (مرحله ۲ و ۴)، تعداد زیادی thread بر روی جی پی یو اجرا می شود.



(شکل-۱): مدل برنامه نویسی کودا-سی.
(Figure-1): CUDA-C programming model.

مجموعه تمام threadهایی که توسط یک تابع کرنل ایجاد می شود، گرید^۸ نامیده می شود. هنگامی که اجرای تمام threadهای یک کرنل پایان یابد، اجرای برنامه بر روی میزبان (سی پی یو) آغاز شده و تا زمان اجرای کرنل بعدی ادامه می یابد.

در بسیاری از کاربردها حجم بالای داده، زمان پردازش نرم افزارهای موجود را در رایانه های سری بسیار طولانی می کند. همچنین بسیاری از این نرم افزارها پدیده های فیزیکی موجود در دنیای واقعی را مدل می کنند. به عنوان نمونه، تصاویر و فریم های ویدئویی، نمایش لحظه ای دنیای واقعی هستند که در آن بخش های مختلف تصویر، پدیده های فیزیکی مستقل را به طور همزمان نشان می دهند. چنین استقلال، پایه ی موازی سازی داده ها است. یک مثال ساده برای درک موازی سازی داده ها، ضرب دو ماتریس بسیار بزرگ در یکدیگر است که در آن هر درایه ماتریس حاصل ضرب، به صورت مستقل از سایر درایه ها قابل محاسبه است. به عبارت دیگر عملیات ریاضی لازم را برای محاسبه تمام درایه های ماتریس خروجی به صورت موازی می توان انجام داد؛ بنابراین ضرب دو ماتریس با ابعاد بالا قابلیت زیادی در موازی سازی داده دارد. موازی سازی داده در کاربردهای واقعی می تواند بسیار پیچیده تر باشد.

۲-۲- واحد پردازنده گرافیکی با معماری کودا

جی پی یو یک پردازنده موازی است که برای کاربردهای گرافیکی ابداع شده است؛ اما در سال های اخیر به عنوان یک پردازنده عمومی نیز به کار گرفته می شود. تفاوت عمده و ساختاری بین جی پی یو و سی پی یو^۱ در این است که معماری سخت افزاری جی پی یو تعداد بیش تری آل یو^۲ را دربردارد و در مقابل، اجزای کمتری برای کش^۳ و واحد کنترل دارد. این امر باعث می شود تا جی پی یو قابلیت محاسباتی بالایی داشته باشد. به طور خاص، جی پی یو توانمندی بیش تری برای اجرای موازی محاسبات ریاضی، که در آن تعداد زیادی عملیات مشابه، روی تعداد زیادی داده انجام می شود، دارد [22].

همان طور که بیان شد، نسل های اولیه جی پی یو تنها قادر به اجرای محاسبات گرافیکی بودند؛ اما از اواخر سال ۲۰۰۷، معماری کودا توسط شرکت انویدیا^۴ به منظور بهره گیری از قابلیت های محاسباتی جی پی یو در سایر

⁵ CUDA C

⁶ Heterogeneous computing

⁷ Launch

⁸ Grid

¹ Central Processing Unit (CPU)

² Arithmetic Logic Unit (ALU)

³ Cache

⁴ NVIDIA

۳-۲- آماره آشکارساز فیشر

در پردازش بلادرنگ سیگنال‌های صوتی، نه تنها آشکارسازی باید به اندازه کافی سریع باشد، بلکه باید روی داد به صورت خودکار و بدون نیاز به مفسر تشخیص داده شود. یک راه برای انجام این کار استفاده از آشکارساز آماری فیشر است که به‌طور گسترده در این حوزه استفاده می‌شود. آشکارساز فیشر از هم‌دوسی بین سیگنال‌های دریافتی توسط حس‌گرهای مختلف در یک آرایه صوتی بهره می‌گیرد. در ادامه به معرفی مختصر این آشکارساز می‌پردازیم. فرض کنیم که یک آرایه متشکل از N حس‌گر، تغییرات فشار جوی را در بازه زمانی $t = 1, \dots, T$ اندازه‌گیری می‌کند، سیگنال‌های ثبت‌شده توسط حس‌گرها در یک ماتریس ذخیره می‌شوند، به‌گونه‌ای که هر سطر شامل سیگنال یک حس‌گر بوده و هر ستون، نمونه‌های زمانی سیگنال‌های حس‌گرها را در یک زمان مشخص نشان می‌دهد. در این صورت یک ماتریس $N \times T$ به‌دست می‌آید. ثبت‌های صورت‌گرفته در محور زمان پنجره‌گذاری می‌شوند و پردازش آرایه‌ای بر روی پنجره‌ها، به‌طور جداگانه صورت می‌گیرد؛ این بازه‌های زمانی بین^۱ نامیده می‌شوند. متوسط هر ستون از این ماتریس به‌صورت زیر نمایش داده می‌شود [7]:

$$\bar{x}_t = \frac{1}{N} \sum_{n=1}^N x_{nt} \quad (1)$$

که در آن x_{nt} نمونه به‌صفت متناظر با زمان t و حس‌گر m ام است. داده‌های ثبت‌شده (حاوی نوفه) به‌صورت زیر مدل می‌شوند:

$$x_{nt} = \bar{x}_t + \epsilon_{nt} \quad (2)$$

که در آن ϵ_{nt} انحراف هر نمونه از مقدار متوسط است. نوفه در این مدل به‌صورت یک متغیر گوسی با متوسط صفر و واریانس σ^2 در نظر گرفته می‌شود. متوسط کل نمونه‌ها عبارت است از:

$$\bar{x} = \frac{1}{NT} \sum_{t=1}^T \sum_{n=1}^N x_{nt} \quad (3)$$

از آنجایی که انحراف هر ستون از متوسط کل را به‌صورت $\alpha_t = \bar{x}_t - \bar{x}$ می‌توان به‌دست آورد، لذا در این حالت مدل داده به‌صورت زیر خواهد بود:

¹ Bin

$$x_{nt} = \bar{x} + \alpha_t + \epsilon_{nt} \quad (4)$$

با توجه به آنچه در بالا ذکر شد، تغییرات کل ثبت‌ها عبارت است از:

$$V_T = \sum_{t=1}^T \sum_{n=1}^N (x_{nt} - \bar{x})^2 \quad (5)$$

$$V_T = V_W + V_B, \quad (6)$$

$$V_W = \sum_{t=1}^T \sum_{n=1}^N (x_{nt} - \bar{x}_t)^2 \quad (7)$$

$$V_B = N \sum_{t=1}^T (\bar{x}_t - \bar{x})^2 \quad (8)$$

که در آن V_B معیاری از توان سیگنال هدف و V_W معیاری از توان نوفه در سیگنال ثبت شده است. با توجه به این که آشکارساز فیشر یک آشکارساز آماری است، توجه به توزیع آماری متغیرهای V_W و V_B بسیار مهم است. می‌توان نشان داد که امید ریاضی این متغیرها به‌صورت زیر است:

$$E(V_W) = T(N-1)\sigma^2 \quad (9)$$

$$E(V_B) = (T-1)\sigma^2 + N \sum_{t=1}^T \alpha_t^2 \quad (10)$$

که در آن σ^2 واریانس نوفه در هنگام ثبت سیگنال‌های حس‌گرهای مختلف است. با توجه به روابط بالا، تخمینی بدون بایاس از واریانس است. این تخمین در شرایط H_0 و H_1 معتبر است. در مقابل عبارت $\frac{V_B}{T-1}$ تنها در شرایط H_0 که α_t صفر است؛ تخمینی بدون بایاس از واریانس است. با استفاده از دو رابطه بالا نسبت فیشر به‌صورت تقسیم دو واریانس تعریف می‌شود:

$$F = \frac{V_B/(T-1)}{V_W/T(N-1)} \quad (11)$$

روشن است در صورت صحیح بودن فرض H_0 مقدار خروجی آشکارساز، به‌طور تقریبی برابر یک و در غیر این صورت بیشتر از یک خواهد بود. با ساده‌سازی رابطه بالا آزمون فرض به‌صورت زیر در می‌آید:

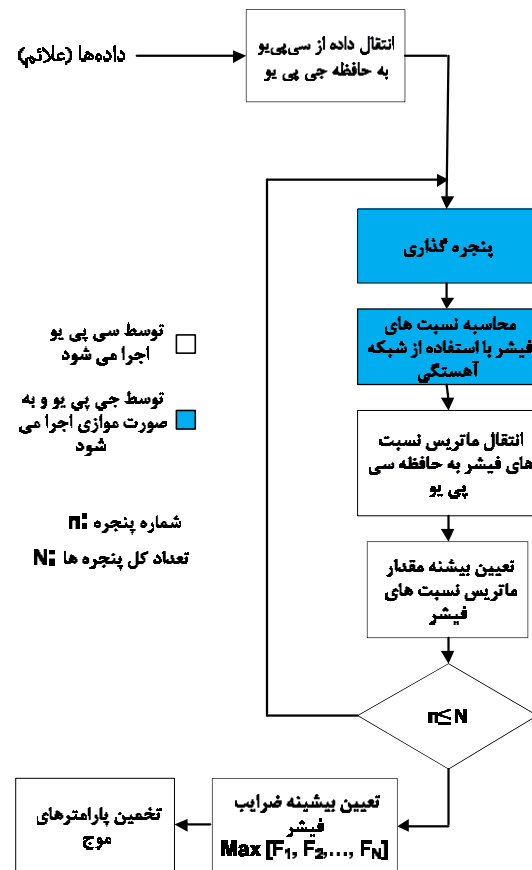
$$F = \frac{T(N-1) \sum_{t=1}^T (\sum_{n=1}^N x_{nt})^2 - \frac{1}{T} (\sum_{t=1}^T \sum_{n=1}^N x_{nt})^2}{N(T-1) \sum_{t=1}^T \sum_{n=1}^N x_{nt}^2 - \frac{1}{T} (\sum_{t=1}^T (\sum_{n=1}^N x_{nt})^2)} \quad (12)$$

با توجه به نحوه تعریف V_W و V_B می‌توان نشان داد، توزیع احتمال آشکارساز فیشر در شرایط H_0 ، یک توزیع ف-مرکزی با پارامترهای $v_1 = T-1$ و $v_2 = T(N-1)$ و

در شرایط H_1 ، یک توزیع-ف غیرمرکزی با پارامتر غیرمرکزی $\lambda_{nc} = v_1 \cdot snr$ است.

۳- روش پیشنهادی پیاده‌سازی الگوریتم فیشر

برای درک بهتر روش پیشنهادی پیاده‌سازی الگوریتم فیشر، روند اجرا در (شکل-۲) نمایش داده شده است. همان‌طور که مشاهده می‌شود، الگوریتم با انتقال داده‌ها از سی‌پی‌یو به حافظه سراسری سی‌پی‌یو آغاز و پس از این انتقال مراحل بعدی در جی‌پی‌یو انجام می‌شود؛ در مرحله پنجره‌گذاری، مجموعه داده‌ها به پنجره‌های زمانی با طول معین تقسیم و در هر پنجره، نسبت فیشر به‌ازای تمامی بردارهای کندی محاسبه و پارامترهای سیگنال (سرعت ظاهری و زاویه سمت‌پشتی) برای بردار کندی متناظر با بزرگ‌ترین نسبت فیشر، تعیین می‌شوند.



(شکل-۲): روندنما و مراحل پیاده‌سازی الگوریتم فیشر.
(Figure-2): Flowchart and implementation steps of Fisher algorithm.

لازم به توضیح است که در روش پیشنهادی، محاسبه نسبت فیشر به‌ازای بردارهای کندی مختلف به‌صورت موازی انجام می‌شود. به‌دلیل آن که تعداد زیادی بردار کندی وجود

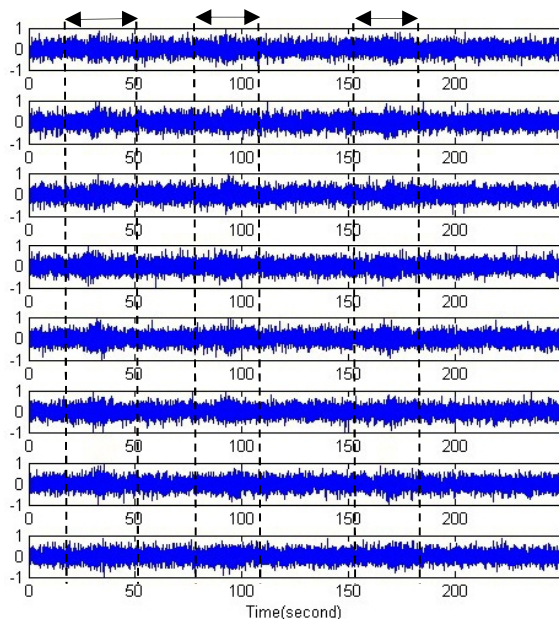
دارد، انتظار می‌رود که با موازی‌سازی پردازش، زمان اجرای الگوریتم به‌طور قابل توجهی کاهش یابد. نسبت فیشر متناظر با هر بردار کندی، در یک ماتریس ذخیره می‌شود. پس از تکمیل این ماتریس توسط جی‌پی‌یو، نتایج حاصل دوباره به سی‌پی‌یو بازگردانده می‌شود. در ادامه، بیشینه این نسبت‌ها توسط سی‌پی‌یو محاسبه می‌شود و پس از آن که بیشینه نسبت فیشر به‌ازای تمامی پنجره‌ها محاسبه شد، به تعداد پنجره‌ها ضرب خواهیم داشت، $[F_1, F_2, \dots, F_N]$. بیشینه این مقادیر مربوط به پنجره‌ای است که بیشترین توان سیگنال منبع صوت را در خود جای داده است و بیش‌ترین هم‌بستگی را بین داده‌های ثبت‌شده توسط حس‌گرهای آرایه دارد. بردار کندی متناظر با این نقطه، سرعت ظاهری و راستای منبع صوت را تخمین می‌زند.

مهم‌ترین پارامترهای مورد استفاده در پیاده‌سازی الگوریتم فیشر عبارتند از: آستانه نسبت فیشر، تفکیک‌پذیری نمونه‌برداری در شبکه بردار کندی، طول پنجره‌های زمانی، مقدار هم‌پوشانی پنجره‌ها و نرخ نمونه‌برداری. مقدار آستانه (ترشولد) آشکارساز فیشر توسط احتمال هشدار غلط^۱ قابل تحمل در سامانه محاسبه می‌شود. تعیین آستانه در گام‌های اولیه نیاز به نظرات محققین خبره در این حوزه دارد تا با استفاده از آن و توزیع F^2 آشکارساز فیشر، آستانه مناسب مشخص شود. تفکیک‌پذیری نمونه‌برداری از شبکه کندی به‌طور مستقیم بر تفکیک‌پذیری جستجوی سرعت و راستای موج تخمینی اثر می‌گذارد؛ از طرفی هرچه شبکه کندی با تفکیک‌پذیری بالاتری نمونه‌برداری شود، بار محاسباتی آشکارسازی افزایش می‌یابد. انتخاب طول پنجره‌های زمانی مناسب، نقش مؤثری در خروجی آشکارساز فیشر در هر پنجره دارد. اگر پنجره زمانی بسیار کوچک در نظر گرفته شود، به‌طوری‌که مقدار آن کمتر از زمان سیر موج از حس‌گر مرجع تا حس‌گر n باشد، در این‌صورت بین این دو حس‌گر در پنجره موجود، هم‌بستگی وجود نخواهد داشت و نمی‌توان از فزونی اطلاعات این حس‌گرها استفاده کرد؛ بنابراین در نظر گرفتن پنجره‌های زمانی بسیار کوچک به تشخیص یک بردار کندی اشتباه منجر خواهد شد. از سوی دیگر هرچه پنجره‌های زمانی بزرگ‌تر باشد، تغییرات مرحله در آن‌ها بیشتر است و هم‌بستگی بین سیگنال حس‌گرها کاهش

¹ False alarm (FA)

² F distribution

می شود. این چینش به دلیل وجود تقارن نسبی نسبت به راستاهای مختلف انتخاب شده است. پاسخ آرایه مربوط به این چینش به ازای بسامدهای ۱ و ۲ هرتز در شکل (۵) نمایش داده شده است. دو دایره رسم شده در این شکل محدوده پاسخ آرایه برای بسامدهای ۱ و ۲ هرتز را نشان می دهند. توجه به این نکته ضروری است که طراحی چیدمان آرایه نقش مهمی در عملکرد الگوریتم دارد. این چیدمان باید بر اساس بازه بسامدی پردازش و تفکیک پذیری زاویه ای مورد نیاز طراحی شود.



(شکل-۳): سیگنال شبیه سازی شده حس گر های آرایه با سه مؤلفه از سه راستای ۹۰، ۶۰ و ۱۲۰ درجه. موقعیت زمانی حضور سیگنال منبع در هر حس گر به صورت تقریبی با خط چین نمایش داده شده است.

(Figure-3): Simulated signals of array sensors from three different directions of 90, 60, and 120 degrees. Approximate time-domain locations of source signals are presented by dotted lines.

شبکه کندی به کار برده شده در این پژوهش در شکل (۶) مشاهده می شود. هر برداری کندی نماینده یک سرعت ظاهری و جهت موج دریافتی است. فرض می شود که موج ورودی از تمام جهات و با سرعت های ظاهری مختلف می تواند وارد شود. برای مقادیر پیش فرض برداری کندی، فرض شده است که نمونه های شبکه به صورت یکنواخت انتخاب شده اند (یعنی فاصله بین هر دو سرعت متوالی، یکسان است). تعداد صد نمونه از هر سطر و ستون این شبکه دریافت شده و در مجموع ده هزار بردار کندی را تشکیل می دهند. البته به دلیل حصول وضوح بیشتر، تمام نمونه ها در شکل (۶) نمایش داده نشده اند. این عدد (۱۰۰×۱۰۰) برابر با

می یابد. در این حالت، بیشینه نسبت فیشر در پنجره ها کاهش می یابد. در بخش مربوط به شبیه سازی مقادیر پارامترهای تنظیم شده ارائه می شود.

(جدول-۱): پارامترهای سیگنال مرجع.

(Table-1): Reference signal parameters.

نام پارامتر	مقدار
بسامد مرکزی	۳ هرتز
بسامد نمونه برداری	۴۰ هرتز
فاکتور تضعیف	۰/۳
هم پوشانی پنجره ها	۸۰٪
طول پنجره	۱۰۲۴ نمونه

۴- نتایج شبیه سازی ها و بحث

در این بخش پس از ارائه مدل سیگنال و چینش آرایه به بررسی نتایج پیاده سازی الگوریتم فیشر بر روی سی پی یو و جی پی یو می پردازیم. در انتها مقایسه سرعت اجرای این الگوریتم در دو معماری سی پی یو و جی پی یو ارائه می شود.

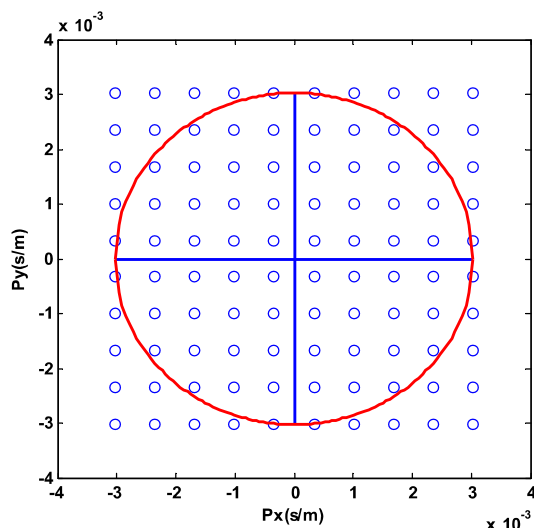
۴-۱- مدل سیگنال و چینش آرایه

مدل سیگنال استفاده شده در شبیه سازی های این بخش از رابطه زیر پیروی می کند:

$$s(t) = ate^{-at}\sin(2\pi ft) \quad (13)$$

که از سه بخش سینوسی، نمایی و خطی تشکیل شده است. f بسامد مرکزی سیگنال و a فاکتور تضعیف نام دارد. نرخ تضعیف سیگنال مرجع، توسط پارامتر a قابل تنظیم است. پارامترهای سیگنال مرجع شبیه سازی شده در (جدول-۱) نمایش داده شده است. به منظور شبیه سازی سیگنال خروجی هر حس گر، سه منبع صوت در راستاهای ۹۰، ۶۰ و ۱۲۰ درجه به ترتیب با سرعت های انتشار موج ۳۶۰، ۳۴۰ و ۳۹۰ متر بر ثانیه در نظر گرفته شده است. لازم به ذکر است که منظور از سرعت انتشار، سرعت ظاهری انتشار موج نسبت به آرایه است. با توجه به زوایا و سرعت های انتخاب شده و همین طور موقعیت مکانی حس گر ها، سیگنال خروجی هر یک از حس گر ها به ازای نسبت سیگنال به نوفه ۵- دسی بل به صورت شکل (۳) در می آید. موقعیت زمانی حضور سیگنال های منبع (به ازای سه راستای ذکر شده) به طور تقریبی توسط خط چین نمایش داده شده است.

آرایه صوتی در نظر گرفته شده در این مقاله، متشکل از ۸ حس گر است. پیکربندی آرایه در شکل (۴) مشاهده



(شکل-۶): نمایش شبکه کندی. نقاط داخل دایره سرعت‌های بیش‌تر از ۳۳۰ متر بر ثانیه را مشخص می‌کند.

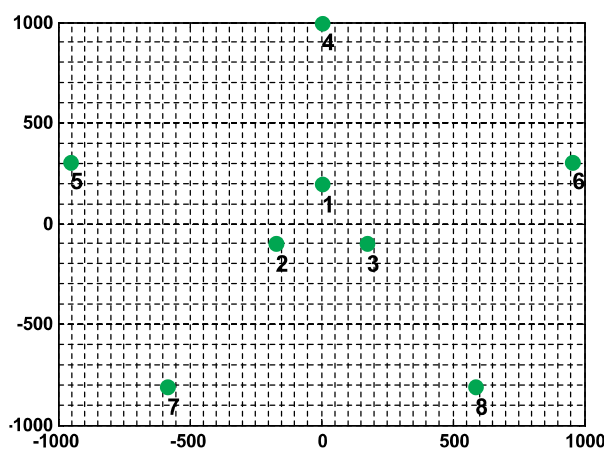
(Figure-6): Slowness-network illustration. The points inside the circle represent the source wave velocities higher than 330 m/s.

پردازنده مورد استفاده در این پژوهش، یک سی‌پی‌یو چهار هسته‌ای i5 شرکت اینتل با بسامد ساعت ۳ گیگاهرتز است. از آنجایی که سرعت اجرای الگوریتم به قابلیت محاسباتی سی‌پی‌یو مورد استفاده نیز بستگی دارد، زمان اجرای الگوریتم با استفاده از دو پردازنده گرافیکی متفاوت یکی با مدل Geforce GT 630، قابلیت محاسباتی ۲/۱ و تعداد کل هسته ۹۶ (جی‌پی‌یو شماره ۱) و دیگری Geforce GTX 650 TI Boost با قابلیت محاسباتی ۳/۰ و تعداد کل هسته ۷۶۸ (جی‌پی‌یو شماره ۲) به‌دست آمده و مقایسه شده است. در شبیه‌سازی‌های این قسمت از نرم‌افزار Visual Studio 2010 و CUDA toolkit 5.0 استفاده شده است. در ادامه نتایج پیاده‌سازی آشکارساز فیشر بر روی سی‌پی‌یو و جی‌پی‌یو تشریح می‌شود.

۴-۲- نتایج تخمین پارامتر موج

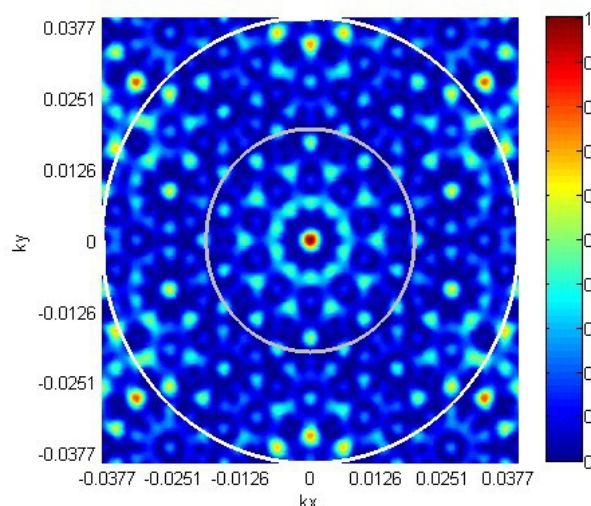
پس از شبیه‌سازی سیگنال‌های دریافتی در محل حس‌گرها، الگوریتم یک بار بدون اعمال پردازش موازی و بار دیگر با اعمال پردازش موازی با استفاده از جی‌پی‌یو پیاده‌سازی شد. در این بخش نتایج پیاده‌سازی الگوریتم فیشر بر روی سی‌پی‌یو و جی‌پی‌یو ارائه می‌شود. در شکل (۷) و شکل (۸) خروجی آشکارساز فیشر به‌ترتیب بر روی سی‌پی‌یو و جی‌پی‌یو نمایش داده شده است. هر یک از این شکل‌ها سه

تعداد کل رشته‌های ایجادشده در کرنل موازی جی‌پی‌یو است. دایره قرمز در شکل (۶) مجموعه بردارهای کندی با سرعت ۳۳۰ متر بر ثانیه را نشان می‌دهد. با توجه به اینکه کمینه سرعت ظاهری موج ۳۳۰ متر بر ثانیه فرض می‌شود، جستجوی شبکه کندی تنها شامل نقاط داخل این دایره می‌شود. با افزایش تعداد نمونه‌های بردار کندی در داخل دایره می‌توان دقت الگوریتم را افزایش داد. این مسئله زمان پردازش را افزایش می‌دهد. به‌عبارت دیگر، با به‌کارگیری پردازش موازی دقت الگوریتم را بهبود می‌توان داد و در عین حال، پردازش نیز به‌صورت بلادرنگ انجام شود.



(شکل-۴): پیکربندی آرایه صوتی (فواصل بر حسب متر).

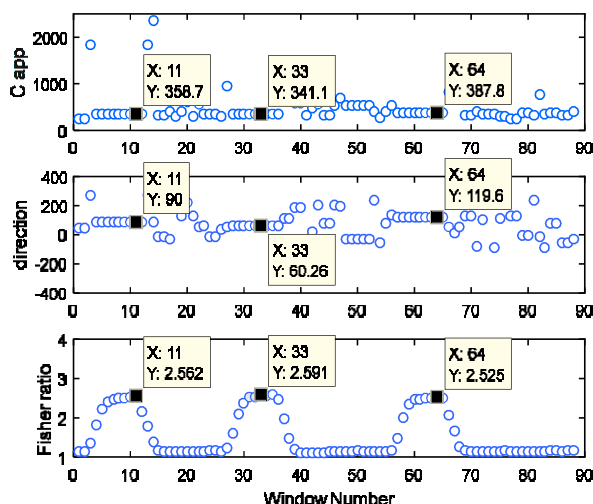
(Figure-4): Acoustic array configuration (distances are in meters).



(شکل-۵): پاسخ آرایه شکل (۴) به‌ازای

بسامدهای ۱ و ۲ هرتز.

(Figure-5): Array response for the frequencies of 1 and 2 Hz.



(شکل - ۸): نتیجه آشکارساز فیشر با استفاده از جی‌پی‌یو شماره ۱ به‌ازای سیگنالی با سه مؤلفه از سه راستای ۹۰، ۶۰ و ۱۲۰ درجه و سرعت‌های ۳۶۰، ۳۴۰ و ۳۹۰ متر بر ثانیه.

(Figure-8): The results of Fisher detector implementation using the GPU no.1 for a signal received from three sources with directions 90, 60, and 120 with velocities 340, 360, and 390 m/s, respectively.

(جدول ۲-۱) مقایسه پارامترهای تخمین زده‌شده توسط سی‌پی‌یو و جی‌پی‌یو.

(Table-2): Comparison between the estimated parameters from CPU and GPU implementations.

پارامتر	سی‌پی‌یو	جی‌پی‌یو
راستای تخمینی (درجه)	119.30	119.6
	60.48	60.26
	90.63	90.00
سرعت تخمینی (متر بر ثانیه)	390.2	387.8
	342.5	341.1
	359	358.7

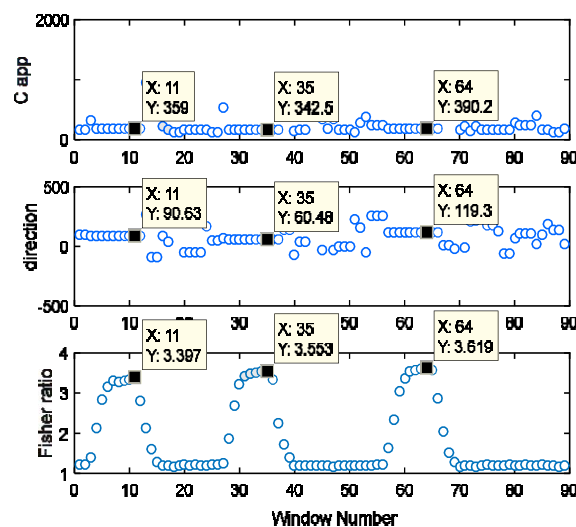
۳-۴- مقایسه سرعت پیاده‌سازی بر روی

سی‌پی‌یو و جی‌پی‌یو

در این بخش به مقایسه سرعت پیاده‌سازی الگوریتم فیشر بر روی سی‌پی‌یو و جی‌پی‌یو می‌پردازیم. جدول (۳) نتایج مقایسه زمان لازم برای اجرای الگوریتم به‌ازای شبکه‌های کندی با ابعاد 50×50 ، 100×100 ، 150×150 ، 200×200 و 250×250 را نشان می‌دهد. همان‌گونه که انتظار می‌رود با افزایش تعداد نمونه‌های شبکه کندی حجم بیش‌تری از محاسبات به‌صورت موازی انجام شده و در نتیجه نسبت بهبود سرعت جی‌پی‌یو به سی‌پی‌یو افزایش می‌یابد. در این جدول نتایج به‌ازای هر دو جی‌پی‌یو ذکر شده در قبل ارائه شده است. با توجه به این که ظرفیت اجرای موازی

نمودار دارد. منحنی پایین، خروجی آماره آزمون^۱ آشکارساز فیشر را به‌ازای پنجره‌های زمانی مختلف نمایش می‌دهد. پنجره‌های متوالی دارای هم‌پوشانی ۸۰٪ هستند. منحنی وسط و بالا به‌ترتیب راستا و سرعت تخمین زده‌شده در هر پنجره زمانی را نشان می‌دهد.

همان‌طور که در منحنی‌های مربوط به نسبت فیشر (منحنی پایین) در شکل (۷) و شکل (۸) مشاهده می‌شود، الگوریتم فیشر حضور سه سیگنال را در پنجره‌های زمانی با شماره‌های تقریبی ۱۱، ۳۵ و ۶۴ در سی‌پی‌یو و ۶۴ و ۳۵ و ۱۱ در جی‌پی‌یو مشاهده می‌کند. مقادیر راستا و سرعت تخمین زده‌شده با هر یک از این سه پنجره در جدول (۲) ارائه شده است. همان‌طور که مشاهده می‌شود، پارامترهای تخمین زده‌شده توسط جی‌پی‌یو با دقت قابل قبولی، تخمین‌های محاسبه‌شده توسط سی‌پی‌یو را دنبال می‌کنند که پیاده‌سازی صحیح کد را نشان می‌دهد. شایان ذکر است که تفاوت اندک میان مقادیر تخمینی توسط سی‌پی‌یو و جی‌پی‌یو، ناشی از مواردی هم‌چون دقت متفاوت اعداد با ممیز-شناور^۲ در محاسبات انجام‌شده توسط سی‌پی‌یو و جی‌پی‌یو و هم‌چنین نوفه کانال تصادفی تولیدی در شبیه‌سازی‌ها است.



(شکل - ۷): نتیجه آشکارساز فیشر با استفاده از سی‌پی‌یو به‌ازای سیگنالی با سه مؤلفه از سه راستای ۹۰، ۶۰ و ۱۲۰ درجه و سرعت‌های ۳۶۰، ۳۴۰ و ۳۹۰ متر بر ثانیه.

(Figure-7): The results of Fisher detector implementation using CPU for a signal received from three sources with directions 90, 60, and 120 with velocities 340, 360, and 390 m/s, respectively.

¹ Test Statistic

² Floating-point

۵- نتیجه گیری

آشکارساز فیشر یکی از مهم ترین ابزارهای پردازشی سیگنال های بسامد پایین در حوزه تخمین پارامترهای موج (راستا و سرعت) است. در این مقاله با استفاده از پردازش موازی مبتنی بر جی پی یو به پیاده سازی آشکارساز فیشر پرداختیم. نتایج شبیه سازی ها نشان می دهد که با استفاده از جی پی یو سرعت اجرای این الگوریتم را تا حدود پنجاه برابر نسبت به سی پی یو می توان افزایش داد. البته این بهبود با استفاده از جی پی یوهای با قدرت محاسبات بالاتر قابل ارتقا است. نتایج بیان گر آن است که پردازش موازی مبتنی بر جی پی یو امکان تخمین پارامترهای موج و آشکارسازی سیگنال را (با استفاده از الگوریتم های موجود در این حوزه) به صورت بلادرنگ تسهیل می کند.

6-References

۶- مراجع

- [1] V. L. Zimmer, N. Sitar, "Detection and location of rock falls using seismic and infrasound sensors," *Engineering Geology*, vol. 193, pp. 49-60, Apr. 2015.
- [2] R.D. Costley, W. G. Frazier, et al., "Frequency-wavenumber processing for infrasound distributed arrays," *Journal of Acoustical Society of America*, vol. 134, no. 4, pp.EL307-EL311, 2013.
- [3] S. Havens, H.-P. Marshall, et al., *Real-Time Avalanche Detection for High Risk Areas*, Research Report, Transportation Department, Idaho University, Dec. 2014, available online at: <https://itd.idaho.gov/highways/research/archived/reports/RP219Final12312014.pdf>.
- [4] W. W. Arrasmith, E. R. Coots, J. V. Olson, and E. A. Skowbo, "Analyzing infrasound and seismic signals emanating from a waterborne system using canonical modeling and analysis methods", *International Journal of Modeling and Optimization*, vol. 4, no. 3, pp. 176-181, Jun. 2014.
- [5] M. Charbit, I. Che, and A. Le Pichon, "Asymptotic distribution of GLRT versus Fisher distribution for infrasonic detection," *Geophysical Research Abstracts*, vol. 15, EGU2013-3690, 2013.
- [6] J. Park, B. W. Stump, C. Hayward, Detection of regional infrasound signals using array data: testing, tuning and physical interpretation, *Journal of Acoustical Society of America*, vol. 140, no. 1, pp. 240-259, Jul. 2016.

محاسبات در سخت افزارهای جی پی یو محدود است، می توان مشاهده کرد که با افزایش بیش تر ابعاد شبکه کندی نرخ افزایش سرعت کاهش می یابد. در جدول (۴) تأثیر طول پنجره در افزایش سرعت پیاده سازی بررسی شده است. در این جدول زمان اجرا به ازای پنجره با طول های ۵۱۲، ۲۵۶ و ۱۰۲۴ نمونه ارائه شده است. همان طور که مشاهده می شود، افزایش طول پنجره تغییر محسوسی در سرعت اجرای الگوریتم توسط جی پی یو ندارد؛ دلیل این امر این است که اگرچه با کاهش طول پنجره، تعداد تکرار حلقه موجود در شکل (۲) افزایش می یابد؛ ولی محاسبات مورد نیاز برای پردازش داده های هر پنجره به همان نسبت کاهش می یابد؛ بنابراین با تغییر طول پنجره سرعت پیاده سازی الگوریتم توسط جی پی یو به طور تقریبی ثابت باقی می ماند. این در حالی است که با کاهش طول پنجره تعداد تکرار جستجوی شبکه کندی افزایش یافته و زمان اجرای برنامه توسط سی پی یو بیشتر می شود.

(جدول-۳): نتایج مقایسه زمان لازم برای پیاده سازی الگوریتم فیشر بر روی سی پی یو و جی پی یو به ازای طول پنجره ۱۰۲۴ نمونه و هم پوشانی ۸۰٪ (اعداد بر حسب میلی ثانیه است).

(Table-3): Results of time-consumptions measurement for the CPU and GPU implementations of Fisher detector with window length of 1024 samples and overlap rate of 80% (times are in milliseconds).

ابعاد شبکه کندی	زمان سی پی یو	زمان جی پی یو شماره ۱	نسبت افزایش سرعت ۱	زمان جی پی یو شماره ۲	نسبت افزایش سرعت ۲
50×50	18.5	3.9	4.8	1.0	18.7
100×100	74.1	8.5	8.7	2.2	33.7
150×150	164.9	16.2	10.3	4.2	39.8
200×200	299.2	27.7	10.7	6.5	45.8
250×250	460.3	42.4	10.9	9.5	48.7

(جدول-۴) نتایج زمانی پیاده سازی الگوریتم فیشر روی سی پی یو و جی پی یو به ازای پنجره با طول های مختلف (ابعاد شبکه کندی ۱۰۰×۱۰۰ می باشد).

(Table-4): Results of time-consumptions measurement for the CPU and GPU implementations of Fisher detector with different window lengths (for slowness network size of 100×100).

طول پنجره	زمان سی پی یو	زمان جی پی یو شماره ۱	نسبت افزایش سرعت ۱	زمان جی پی یو شماره ۲	نسبت افزایش سرعت ۲
L = 256	139.4	9.5	14.7	2.8	49.8
L = 512	102.5	8.5	12.1	2.5	41
L = 1024	74.1	8.6	8.6	2.2	33.7

algorithm on GPUs using CUDA”, *Procedia Computer Science*, vol. 3, pp. 396-400, 2011.

[21] L. Mussi, F. Daolio, S. Cagnoni, “Evaluation of parallel particle swarm optimization algorithms within the CUDA™ architecture”, *Information Sciences*, pp. 4642-4657, 2011.

[22] D. B. Kirk, W.H. Wen-Mei, *Programming Massively Parallel Processors: A Hands-on Approach*, 2nd Edition, Morgan Kaufmann, 2011.



حامد صادقی مدرک کارشناسی خود را

در رشته الکترونیک در سال ۱۳۸۴ از دانشگاه شاهد اخذ کرد؛ سپس دوره‌های کارشناسی ارشد و دکترای خود را در دانشگاه تربیت مدرس از سال ۱۳۸۷ تا

۱۳۹۲ به اتمام رساند. ایشان در زمینه‌های مخابرات بی‌سیم، پردازش سیگنال‌های صوتی و تئوری آشکارسازی سیگنال، پژوهش‌های خود را پیش می‌برد.

نشانی رایانامه ایشان عبارت است از:

h.sadeghi.2015@ieee.org



امیر اخوان مدرک کارشناسی مخابرات

خود را در سال ۱۳۸۷ از دانشگاه صنعتی اصفهان، کارشناسی ارشد خود را در سال ۱۳۹۰ از دانشگاه تربیت مدرس و دکترای مهندسی پزشکی خود را در سال ۱۳۹۶ از

دانشگاه صنعتی امیرکبیر دریافت کرد. ایشان هم‌اکنون استادیار دانشگاه صنعتی همدان بوده و پژوهش‌های خود را در زمینه‌های پردازش سیگنال‌های صوتی و سیگنال‌های حیاتی به‌پیش می‌برد.

نشانی رایانامه ایشان عبارت است از:

amir.akhavan@aut.ac.ir

[7] S. J. Arrowsmith, R. Whitaker, S. R. Taylor, “Regional monitoring of infrasound events using multiple arrays: application to Utah and Washington State”, *Geophysical Journal International*, pp. 291-300, Jul. 2008.

[8] S. J. Arrowsmith, R. Whitaker, C. Katz, C. Hayward, “The F-detector revisited: An improved strategy for signal detection at seismic and infrasound arrays”, *Seismological Society of America*, vol. 99, no. 1, pp. 449-453, 2009.

[9] B. Melton, and L. Baily, “Multiple signal correlators,” *Geophysics*, XXII (3), pp. 565-588, 1957.

[10] R. R. Blandford, “An automatic event detector at the Tonto Forest seismic observatory”, *Geophysics*, vol. 39, pp. 633, 1974.

[11] S. Angelis *et al.*, “Detecting hidden volcanic explosions from Mt. Cleveland Volcano, Alaska with infrasound and ground-coupled airwaves,” *Geophysical Research Letters*, vol. 39, 2012.

[12] L. G. Evers and H. W. Haak, “Tracing a meteoric trajectory with infrasound”, *Geophysical Research Letters*, vol. 30, no. 24, Dec. 2003.

[13] Y. Cansi, “An automatic seismic event processing for detection and location: the PMCC method”, *Geophysical Research Letters*, vol. 22, pp. 1021-1024, 1995.

[14] J. Nickolls, W.J. Dally, “The GPU computing era,” *IEEE Micro Magazine*, vol. 30, pp. 56-69, 2010.

[15] H. Chen, S. Saighi, L. Buhry, and S. Renaud, “Real-time simulation of biologically realistic stochastic neurons in VLSI,” *IEEE Transactions on Neural Networks*, vol. 21, no. 9, pp. 1511-1517, Sep. 2010.

[16] S. U. Gjerald, R. Brekken, T. Hergum, J. D’hoog “Real-time ultrasound simulation using the GPU,” *IEEE Transactions on Ultrasonic Ferroelectrics and Frequency Control*, vol. 59, pp. 885-892, 2012.

[17] Y. Dai, *et al.*, “Real-time visualized freehand 3D ultrasound reconstruction based on GPU,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, pp. 1338-1345, 2010.

[18] C. Richter, S. Schops, and M. Clemens, “GPU acceleration of finite difference schemes used in coupled electromagnetic/ thermal field simulations”, *IEEE Transactions on Magnetics*, vol. 49, no. 5, pp. 1649-1652, May 2013.

[19] W. Rodrigues, *et al.*, “Accelerating atomistic calculation of quantum energy eigenstates on graphic cards”, *Computer physics Communications*, pp. 2510-2518, May 2014.

[20] A. Artu, “Parallel wavelet-based clustering

